Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

A Multi-Objective Optimized Service Level Agreement Approach Applied on a Cloud Computing Ecosystem

LEONILDO JOSÉ DE MELO DE AZEVEDO[1], JÚLIO C. ESTRELLA[1], CLÁUDIO F. MOTTA TOLEDO[1], AND STEPHAN REIFF-MARGANIEC[2], (Member, IEEE)

[1]Institute of Mathematics and Computer Sciences (ICMC) – University of São Paulo (USP), São Carlos, SP, Brazil (e-mail: leonildo.azevedo@usp.br, jcezar@icmc.usp.br, claudio@icmc.usp.br)

[2]School of Electronics, Computing and Maths, University of Derby, Derby, UK, (e-mail: sreiffmarganiec@ieee.org)

Corresponding author: Leonildo José de Melo de Azevedo (e-mail: leonildo.azevedo@usp.br).

This work was supported in part by CNPq, CAPES, and FAPESP (processes IDs: 15/11623-4 and 16/14219-2), and another part of this research carried out using the computational resources of the Center for Mathematical Sciences Applied to Industry (CeMEAI) funded by FAPESP (grant 2013/07375-0).

ABSTRACT The cloud ecosystem provides transformative advantages that allow elastically offering on-demand services. However, it is not always possible to provide adequate services to all customers and thus to fulfill service level agreements (SLA). To enable compliance with these agreements, service providers leave the customer responsible for determining the service settings and expect that the client knows what to do. Some studies address SLA compliance, but the existing works do not adequately address the problem of resource allocation according to clients' needs since they consider a limited set of objectives to be analyzed and fulfilled. In previous work, we have already addressed the problem considering a single-objective approach. In that work, we identified that the problem has a multi-objective characteristic since several attributes simultaneously influence the SLA agreement, which can lead to conflicts. This paper proposes a multi-objective combinatorial optimization approach for computational resources provisioning, seeking to optimize the efficient use of the infrastructure and provide the client with greater flexibility in contract closure.

INDEX TERMS Cloud Computing Ecosystem, Metaheuristics, Multi-Objective Optimized, SLA, QoS.

I. INTRODUCTION

The paradigm of cloud computing has brought a major change in the context of how computing resources are currently offered. Resources are offered on demand, allowing to achieve greater flexibility and scalability to meet the user's needs. Companies, universities and governments have chosen to use cloud resources in order to reduce costs usually incurred with in-house infrastructures. Specific popular features for business owners are the combination of low investment in infrastructure and low cost of operation paid for high scalability and ease of access. [37].

The use of the cloud ecosystem has emerged as a Google proposal for all types of Internet users (individuals and companies) [8]. A cloud ecosystem can be defined as a complex system that is composed of independent components that enable cloud services. This ecosystem is composed of living and nonliving objects, e.g., hardware, software, cloud customers, cloud engineers, integrators, and partners, and all of theses components are connected and work together [23].

Examples of available services in the cloud ecosystem are Google Docs [1], Amazon Elastic Compute Cloud and Simple Storage Services [2], Microsoft Windows Azure Platform [3], IBM Smart Business [4], SalesForce.com [5], among others. These address both application specific as well as more fundamental compute resource provision. The lack of overarching standards for the cloud computing model is the subject of ongoing discussion. Currently, each service provider builds their cloud computing services according to their own policy. So despite the many advantages, the cloud ecosystem still

[1]http://docs.google.com
[2]http://aws.amazon.com/ec2
[3]http://www.windowsazure.com
[4]http://www.ibm.com/cloud
[5]http://www.salesforce.com

has some problems mainly related to data confidentiality, scalability, security, and SLA management [35] [29].

The providers usually offer services which can be grouped into three main categories: *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) and *Software as a Service* (SaaS). Through these basic services, each provider defines its business model by organising a computing environment, where virtual components are offered to its clients. These components appear within an interface that implements the computing platform. Although we focus on the IaaS model, the developed methodology could also be apply to the others models.

For a better use and distribution of computing resources, techniques such as virtualization are applied. This technique consists of allowing multiple operating systems to exist on the same physical host, keeping a strong logical isolation between virtualized components [20], and provides better management and fault tolerance [20] as additional advantages. The management needs to consider optimization of computing resources for providers and also looks at the needs of the clients that "consume" such resources.

However, it is necessary to verify the impact generated in the system during the commitment and provision of computational resources to clients. The system should achieve the performance contracted by a client, but might also increase the cost to be paid. Both the impact of the changes made and the response to the customer must be provided quickly. The big challenge is to quantify resources to meet customer needs as accurately and tightly at possible in order to meet the Quality of Service (QoS) in the SLA while minimizing the use of the cloud resources. The former is required to satisfy the customer, the latter to maximise profit for the provider.

In this paper, we propose a novel multi-objective optimization method for the provisioning of resources in clouds. The method considers the trade-off between cost and makespan by applying different services types for different SLAs. NSGA-II is the chosen multi-objective method to generate SLAs within a Pareto frontier. Moreover, the users can choose among acceptable SLAs based on their preferences. The proposed approach is evaluated over Amazon EC2 configurations with the Cloud Sim simulator. The main contribution of our study is to deal with such multi-objective problem by applying a multi-objective method that looks at different kinds of services, focused on the establishment of SLA, aiming for a better trade-off between cost and makespan. The specific novel contributions of the paper are: 1) a Non-dominated Sorting Genetic Algorithm II (NSGAII) applied to find the multiple optimal SLAs through meta heuristics; and 2) a more robust multi-criteria analysis significantly improving on what is achievable with a single objective.

The paper is structured as follow. A literature review is conducted in Section II, which addresses optimization within a cloud ecosystem. In Section III, the problem that will be tackled and solved in this study is defined precisely. Section IV describes the methods employed for the solution of the problem. In Section VI, 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 VII

II. LITERATURE REVIEW

There are several papers in the literature that analyse and propose mechanisms for the management of resources in a cloud environment. The proposal by Amazon for automatic reconfiguration of the infrastructure of its customers is based on monitoring through alerts (*CloudWatch Alarms*) and policies (*Scaling Policies*). The works found in the literature that address the provisioning of resources can be classified as follows: dynamic policies, based on heuristics, multi-criteria and optimization [14].

There are several techniques that aim to optimize resource utilization through task scheduling and workload evaluation [34] [30] [24]. However, these techniques only apply specific heuristics to the problem. They do not take into account any attributes of the SLA, and whether an SLA is being fulfilled or not.

Heuristic-based approaches for SLA assume that a set of heuristics are pre-defined to be applied in some scenarios. These strategies are relatively simple, with several heuristics being developed and added at runtime. For example, in [5] a set set of rules is created and applied in a multilevel heuristics, the rules can apply or not depending on the SLA violation and its level. In another example [7] a multilevel heuristics is also applied, with the objective to monitor virtualized resource usage and to trigger migration actions appropriately to avoid resource starvation. However, this is limited to predicting specific scenarios at a given time. SLA violations may occur in scenarios that are not included in this prediction.

Several works apply heuristics to the resource provisioning problem. Some apply heuristics to the automatic start-up of VMs. [12] applied two heuristics, the "Scheduling Heuristic" and the "Load Balancing Strategy", where the first one provides more VMs in case the VM list in the load balancing strategy is not enough.

Another work, [21], uses a metaheuristic approach to reduce resource utilization to achieve energy savings. For this, a multi-objective version of the EMLS-ONC (Energy-aware Multi-start Local Search algorithm) was proposed to find a Pareto tradeoff between reducing the energy consumption and preserving the VMs performance. However, this work does not make optimum decisions and considers a very limited number of SLAs and clients and only one QoS attribute.

The techniques based on optimization use approaches similar to the heuristic methods. On the other hand, optimization approaches can be used in predictions and reaction to SLA violation, thus these approaches have a higher complexity than heuristic methods, applied to the detection and treatment of SLA violation [14]. In this context, the detection may occur through analysis of the system performance model or occurrence of failures, in order to adjust the capacity of contracted VMs.

The optimization approaches generally use machine learning methods, time series analysis or fault tolerance techniques, amongst others. [13] applied a dynamic bin packing approach to allocate a set of VMs on a set of physical machines (PMs), with the objective to achieve a high utilization on the PMs and at the same time, avoid SLA violation in terms of VM migrations. [19] proposes a cloud resource autoscaling scheme at the IaaS level to web applications, the achieved objective was to reduce the VM cost, however, the SLA violations was not avoided. [32], investigate adaptive approaches for resource allocation and energy management, using measurable data collected in queuing backlogs, request sizes, VM utilization, and request throughput, to associate them with the resource adjustment and power management decisions. However, these methods do not prioritize SLAs and suffer from high complexity and thus take a lot of time to solve specific problems.

Multi-criteria solutions to the resource provisioning problem tend to be decentralized, i.e., evaluating each criteria or situation independently. [36], focuses on the problem of resource management, where the task selection was modeled as a multi-criteria decision making problem. They utilised the IMPROMPTU model for distributed Multiple Criteria Decision Analysis (MCDA), this model distributes the responsibility of resources among 3 autonomous node, (1) one to monitor, (2) one to register undesirable situations, and (3) another one to ensure that the desirable condition on a physical machine is restored. By applying this model it is possible to reduce the resource's fault but was not possible to avoid the SLA violations. A literature review [16] of the multi-criteria decision making approaches for supplier evaluation and selection highlights several further works in this context, however none deals with SLA violations – yet these are crucial as they highlight unsatisfied customer demands.

There are several complex problems within the context of cloud computing that are addressed by solutions that use optimization. [4] surveys VM allocation problems, however, just a few works address the SLA problem and they only consider a very limited number of SLAs (3 to 15), and for the most part only one client run per experiment. [3] is most closely related to our approach. The work has the objective to minimize cost and still guarantee satisfactory performance in order to satisfy SLA and efficient use of resources. However, this work does not have an optimally decision, and does not consider conflicting objectives.

There are other related works that propose mechanisms for resource management in a cloud environment. For example, the problem of allocating virtual machines in a real machine [28], energy saving [4], scaling and load balancing of applications in virtual machines [27], the use of resources aimed at reducing costs, while still guaranteeing a satisfactory performance [6], and ensuring the QoS is in compliance with the SLA.

In order to highlight our contribution Table 1 presents the main features of the related works, wit the following columns:

- **Related work:** reference to the related work addressed;
- **Environment:** the experimental environment, either *Real world* (e.g. a prototype) or *Simulator* (i.e. a simulated experiments in a fictitious environment);
- **SLA:** whether the approach focuses on the SLA to make a decision;
- **Optimization technique:** whether the approach used any optimization technique;
- **QoS:** whether the approach considers the QoS attributes in the resource provisioning;
- **Solutions:** the resource provisioning problem could have many solutions to satisfy the problem in different ways – does the approach lead to more than one?;
- **Multi-objective:** whether the approach treats the problem as a multi-objective problem and/or considers any conflicting objectives.

Problems such as task scheduling and resource provisioning are considered NP-hard [25]. Many problems in this complexity class are solved by integer programming and branch-and-bound approaches [33]. However, these are not suitable to solve decision problems that have continuous adaptation [26].

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 [9]. To solve this problem in the single-objective way, we have chosen several metaheuristics environment. However, the best performance was achieved with the Multi-Population Genetic Algorithm (MPGA). On the other hand, all of those methods look to only one solution, and no-one considered other possibilities. In the real world, multiple criteria rarely have the same weight. For this reason, a multi-objective method is necessary.

III. PROBLEM STATEMENT

Many optimization problems deal with conflicting goals, usually improving the outcome with regard to one goal incurs a worsening for other goals. For example, we could have a bigger house, but it will either cost more or be in a less desirable area. Such problems are classified as Multi-objective Optimization Problem (MOP) and the aim is to find the best trade-off across all criteria.

For cloud executed workloads, it is possible to identify two main conflicting objectives: Makespan and Cost. For instance, if we prioritise to reduce investment when acquiring computer infrastructure, it can lead to lower computational power and, consequently, an increased makespan for scheduled tasks. The graph of Figure 1 illustrates this conflict of objectives.

Makespan and cost are factors that depend on the number of virtual machines contracted by the client. In this paper, three types of Virtual Machines (VMs) are considered: Small, Medium, and Large. These requests were based on the configuration of the m3.medium, m3.large, and m3.xlarge

TABLE 1. Main features of the related works.

Related work	Environment	SLA	Optimization technique	QoS	Solutions	Multi-objective
[Providers, 2018]	Real world	√	–	–	–	–
[30]	Simulator	–	–	–	–	–
[24]	Simulator	–	√	√	–	–
[34]	Simulator	√	–	√	–	–
[12]	Simulator	√	–	–	–	–
[21]	Simulator	√	√	–	–	–
[13]	Simulator	√	√	–	–	–
[36]	Simulator	√	√	–	–	–
[3]	Simulator	√	–	√	–	–
This work	Simulator	√	√	√	√	√

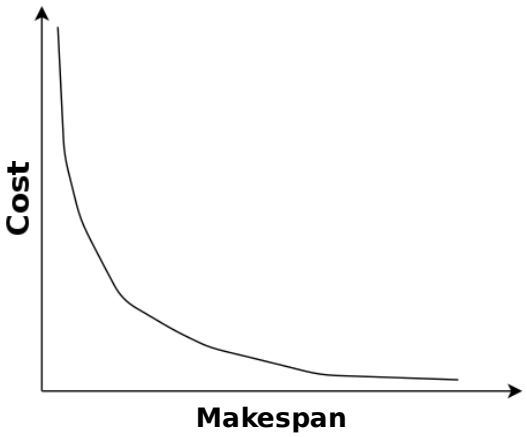

FIGURE 1. The conflicting objectives Makespan of Cost.

applications of Amazon EC2[6] (Other providers operate on similar schemes). A *data center* infrastructure can have a large number of VMs with a wide range of different configurations; the total cost could be considered by the sum of costs for all VMs used, and makespan means the application response time with a specific set of VMs.

- **Cost per hour (Cost/h):** the monetary value defined 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 1 [22]:

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

where,
$Cost(VM_i)$ is the cost of a specific VM;
n is the number of VM instances considered for deployment.

- **Makespan:** 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 [22]:

$$Makespan = \frac{\sum_{i=1}^{i=n} Makespan(VM_i)}{n} \tag{2}$$

where,
$Makespan(VM_i)$ is the response time of the application part in a specific VM;
n is the number of VM instances considered for deployment.

These two QoS attributes allow estimating the minimum and the maximum values for each one. Therefore, the values are set up in the SLA for the client after application of the optimization method.

The management of an SLA is a task composed of several phases, namely negotiation, implementation, monitoring, violation management, reporting, and finalization [14] forming an SLA lifecycle:

- **Negotiation:** define the terms of services and include monetary aspects;
- **Establishment:** requests from clients are assigned to the provider resources;
- **Monitoring:** periodic monitoring of the resources and the status of the execution;
- **Violation management:** monitoring might flag issues with resources or the execution and these need to be addressed and resolved;
- **Reporting and Termination:** provide SLA reports containing detailed information of activities that occurred during service usage;
- **Termination:** a method for parties to the agreement to terminate the SLA.

Figure 2 illustrates the sequence in which these steps are performed [14]. In this paper, we approach the first two

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

phases, i.e. the negotiation and establishment phases. We provide many SLA possibilities where the client can choose which one is best for their requirements and configure the resulting set up of the contracted resources (established in the SLA).

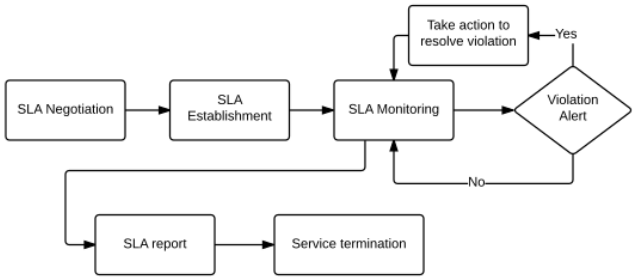

FIGURE 2. SLA management life cylcle [14].

IV. METHODOLOGY

When a mechanism is well designed for resource provisioning in a cloud environment, it is possible to achieve cost savings, better use of available infrastructure, and better performance in the context of variations in demand for services [3]. On the other hand, the provisioning process is not trivial [18]. According to [15], it requires defining better software and hardware configurations to ensure compliance with SLAs and to meet the need to maximize system efficiency and usage.

As identified in the problem statement, the aim is to find the optimal machine configuration to satisfy the SLA. In this context, we propose a multi-objective optimization to the provisioning of resources in cloud environments that considers the Pareto trade-off between cost and makespan by applying different service types for different SLA. The overall aim is to achieve the best balance between makespan and cost for the customer, and the best resource utilization for the provider.

A. ENCODING

The required method must optimize the number and type of virtual machines contracted by the client, identifying the necessary resources for each context precisely. Machines are typically obtained in predefined sizes, such as small or extra large, from common providers. Thus, the representation of a solution (encoding) is defined by (s, m, l) corresponding, respectively, to VM types *small*, *medium* and *large*. This is the set of possible VMs to be contracted, arranged in a vector, where each position of the vector corresponds to the number of machines of a given type as shown in Figure 3.

The clients can set the desired capacity (C^c) as well as the expected makespan and cost per hour (C/h^c). However, it is difficult to meet all requirements without conflict. For example, the capacity request may not meet the desired cost, or it may not give the desired response time. For this reason, the methods described in this Section will be applied in order

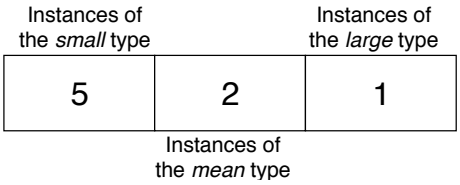

FIGURE 3. Representation of the triple coding (s, m, l).

to find the best set of VMs $(s*, m*, l*)$ that deliver as close as possible to the clients requested cost and makespan.

B. THE OPTIMIZATION METHODS

In this paper, we compares two optimization approaches: a mono-objective and a multi-objective approach. The mono-objective method is MPGA as introduced in our previous work [9] and briefly described in section IV-B1. The novel multi-objective method proposed in this paper is an adaptation of NSGA-II, detailed in section IV-B2, which provides the Pareto trade-off between cost and makespan to optimize different service types for different SLAs.

1) MPGA

Algorithm 1 describes MPGA which is based on the hybrid genetic algorithm proposed in [31].

Algorithm 1 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   s
21: end procedure
```

Each individual represents a possible VM configuration (s, m, l) for the client. $InitializePopulation(P)$ 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 seems small, but aims to reduce the effort for fitness evaluation

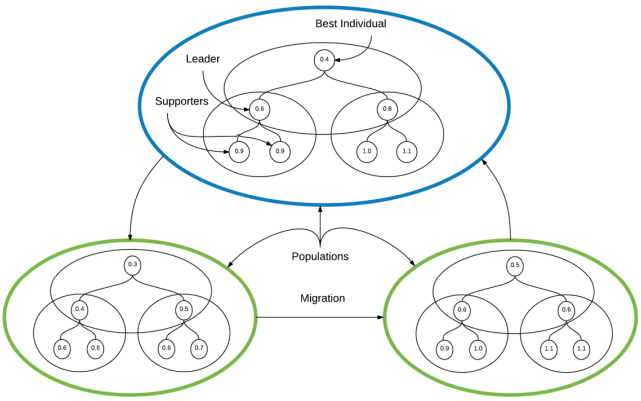

FIGURE 4. Population structure and migration.

$Evalutate(P[i])$. This is crucial as it will execute simulations using CloudSim for each configuration. 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 steps (lines 10-14).

Figure 4 illustrates the population structure. The position of the individuals (nodes) in the clusters indicates their value within the hierarchy. In each cluster, the followers have worse fitness than their leader. Thus, the best individual will be the root in such a hierarchical tree structure, while the worst individuals are at the leaves. $Selection(P[i])$ randomly selects a follower as one parent and its leader as the other parent. The new individual is evaluated next and the procedure $Structure(P[i])$ may include it in the hierarchical structure (line 14) when its fitness is better than that of the worst parent. In this case, $Structure(P[i])$ will also update the positions throughout the tree hierarchy. For instance, if the new individual is also better than the best individual found so far, it will become the root node in the tree.

The evolutionary steps carried out 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, $restartPopulation(P[(imodnPopulation) + 1])$ produces a new population, but maintains the two best individuals identified.MPGA stops when the time limit is reached.

In our previous work, we merged the QoS attributes in order to obtain an objective function. In the current work, each Equation (1 and 2) described in Section III becomes a weighted objective. Thus, the mono-objective function (fitness) that guides the MPGA is as given by Equation 3.

$$MonoFit = \left|\frac{T^c - T^*}{T^*}\right| + \left|\frac{C/h^c - C/h^*}{C/h^*}\right| \quad (3)$$

2) NSGA-II

In our previous work, the objective function employed a normalization of the different objectives. While this obtains a solution capable of establishing the contract from the client's demands, the conflict between the existing objectives may not have been properly addressed when assuming equal weights. Therefore, to evaluate and offer solutions that are even more adherent to the context of the problem, we propose the application of a multi-objective meta-heuristic called NSGA-II.

NSGA-II was proposed by [10] as an improved version of NSGA. The method randomly creates a population (P), with mutation and crossover operators being applied next. Those operators will define a population of offspring (F). After the offspring generation, both the children and their parents are grouped into a set (Q). In this set, the non-dominance comparison is applied. This results in a first group f_1 of non-dominant solutions. The non-dominance comparison is applied again to the remain individuals, generating other groups (f_2, f_3,...f_n). In the last step, the individuals within each group are sorted by decreasing values of their distances. Such distance can be defined, e.g., based on individuals' fitness value. The next population will have the individuals of the best groups with the largest distance measure between them. The distance criterion will ensure diversity of the population. This process is repeated until a stop criterion has been satisfied. In our case, we set an execution time limit as stop criterion. Figure 5 illustrates this procedure.

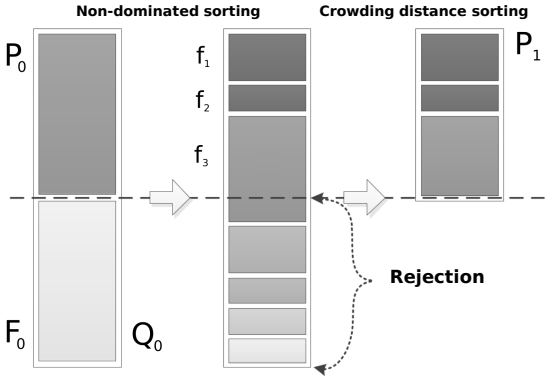

FIGURE 5. NSGA-II illustration (adapted from [10]).

NSAG-II applies a method to order the solutions by non-dominance named *fast-non-dominated-sort*, which calculates the dominance in two steps. First, for all solutions, a degree of dominance (n_p) is calculated based on the number of solutions dominating a solution $p = (s, m, l)$. If the value of n_p is 0, it means that a solution p is not dominated and it will be part of the first set. The second step is to separate the solutions into groups S_p in order of dominance. Therefore, each individual that is added to a set S_p is removed from the population, and the individuals dominated by it have their value of n_p decremented. Step two is repeated until there are no more individuals in the population. In the algorithm 2, these two steps are detailed, where q is another population solution to be compared with p.

After indexing the solutions within the sets by non-dominance, the solutions are sorted by the distance of their

Algorithm 2 *fast-non-dominated-sort*

```
1: for Each p ∈ P do
2:     S_P = ∅
3:     n_p = 0
4:     for q ∈ P do
5:         if p ≺ q then
6:             S_p = S_p ∪ {q}
7:         else
8:             if q ≺ p then
9:                 n_p = n_p + 1
10:            end if
11:        end if
12:    end for
13:    if n_p = 0 then
14:        f_1 = f_1 ∪ {p}
15:    end if
16: end for
17: i = 1
18: while f_i ≠ ∅ do
19:    Q = ∅
20:    for Each p ∈ f_i do
21:        for Each q ∈ S_p do
22:            n_p = n_p − 1
23:            if n_p = 0 then
24:                Q = Q ∪ {q}
25:            end if
26:        end for
27:    end for
28:    i = i + 1
29:    f_i = Q
30: end while
```

fitness values. This distance is obtained by calculating the average distance between a center point i. Figure 6 presents the arrangement of points in relation to this distance.

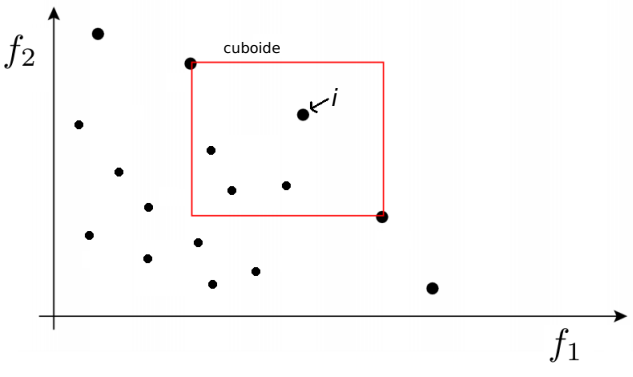

FIGURE 6. Distance between fitness functions (adapted from [10]).

Once the solutions are indexed in the order of non-dominance and following the distance between functions, the whole evolutionary process is repeated until the stop criterion has been met. Line 20 in Algorithm 3 takes care of the ordering by the distance of the fitness values in the last set of f, and line 21 selects the best individual of the last set to remain in the next population.

Algorithm 3 NSGA-II algorithm

```
1: procedure NSGA-II(SLA: C^c, T^c, A^c, C/h^c)
2:     t = 1
3:     InitializePopulation(P_t)
4:     Evaluate(P_t)
5:     repeat
6:         F_t = copy(P_t)
7:         Selection(F_t)
8:         Crossover(F_t)
9:         Mutation(F_t)
10:        Evaluate(F_t)
11:        Structure(F_t)
12:        Q = P ∪ F
13:        f[] = fast-non-dominated-sort(Q)
14:        P_{t+1} = ∅
15:        i = 1
16:        repeat
17:            sort-by-distance(f_i)
18:            P_{t+1} = P_{t+1} ∪ f_i
19:            i = i + 1
20:        until |P_{t+i}| + |f_i| ≤ P_t.tamanho
21:        Odernar(f_i, ≺_{n_p})
22:        P_{t+1} = P_{t+1} ∪ f_i[1 : (N − P_t + 1)]
23:        t = t + 1
24:    until determined time
25:    return P_{t−1}
```

C. OPERATORS

The operators used for mutation and crossover, are the same as proposed in [9] for MPGA. The crossover operator generates a new individual from two parents by applying blx-α (blend alpha crossover) and uniform crossover [11]. 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 [11].

In Figure 7, the offspring (s', m', l') inherits $s' = s^1$,$m' = m^{'}$ 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 in Figure 8. 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.

The mutation operator can be applied if the 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);

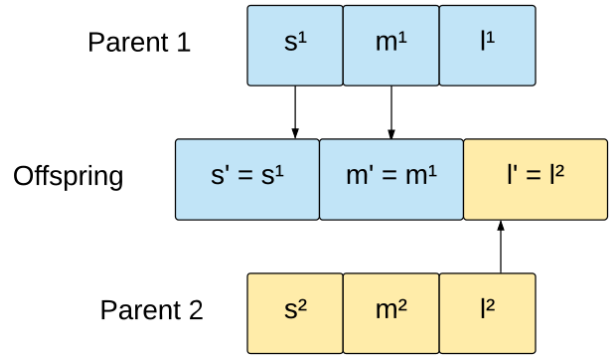

FIGURE 7. Uniform crossover [9].

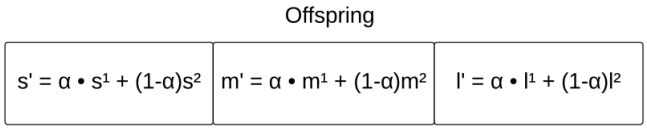

FIGURE 8. Blx-α crossover [9].

- **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).

V. USE CASE

In this section, we present a concrete use case with QoS indicators and related target. The use case will exemplify and later be used to validate our method. Thus, here we show the resources provisioned and detail how the application/system/service is deployed in the modelled cloud.

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 [17]. However, it should be remembered that the clients do not always have the technical knowledge to handle the provisioning of resources, and such task could be burdensome for them. For this reason, solutions as the one presented in this paper are necessary. Our solution intents to ensure the maximum use of computational resources, but leading the clients to get the right amount of and pay a fair price for the services to achieve the required QoS.

To address how the application/system/service is deployed in the cloud modelled, our use case assumes that the customer indicates the application (service) that they would like to deploy in the cloud modelled. The customer will also set the maximum cost that they would like to pay and the QoS required. From such input, our method will return the range of computational resource that best satisfy the customer requirements to reach a satisfactory SLA.

In our use case, we suppose two clients with two benchmark applications: Apache [1] and the Smallpt [2] benchmarks. Apache is an I/O bound application based on a repository of files. Smallpt is a CPU bound application based on image rendering. Each one of these applications can have a different behaviour based on a Cloudlet[7] length variation. For instance, a minimum workload is generated to establish tasks demanding less computational power, then a maximum of computational power is spend solving bound tasks. It is a dynamic system since the workload is generated following the type of service that the client wants to deploy in the cloud.

We are using the most common QoS parameters for SLA, according to [14], in our use case: makespan and cost. They were properly introduced in section III. A further two parameters are also considered: computational capacity of the Virtual Machines (VMs) and the workload. The SLA generator applied here is the same as described in [9], where a Gaussian distribution defines values for the QoS parameters. Table 2 gives some examples of SLAs defined by their QoS indicators.

TABLE 2. SLAs samples.

	VMs (s, m, l)	Makespan (sec.)	Cost/h (U$)	Workload*
SLA 1	X*	100	0,2	80500
SLA 2	X*	150	0,53	100000
SLA 3	X*	180	0,6	125000
SLA 4	X*	200	0,7	170000

*VMs it's the output of the algorithm;
**Cloudlet length: depends on the chosen service.

The infrastructure of the cloud computing ecosystem follows the model adopted by Amazon M3 instances. The M3 instances feature high-frequency Intel Xeon E5-2670 (Sandy Bridge or Ivy Bridge) processors and SSD-based instance storage. Table 3 shows the M3 instances configurations employed by this use case.

TABLE 3. Specification of instances

Instances	Virtual Core	Main Memory (GB)	Disk SSD (GB)	Price/Hour (U$)
m3.medium	1	3.75	1 x 4	0.113
m3.large	2	7	1 x 32	0.225
m3.xlarge	4	15	2 x 40	0.450

In this scenario, the client must indicate the application service to be execute, the desired makespan and the maximum cost that she/he would like to pay for. On the other hand, the optimization algorithms will return the infrastructure range that meets the user requirements or the closest approximation.

[7]The tasks or jobs in CloudSim simulator are called Cloudlets.

VI. COMPUTATIONAL RESULTS

The aim of the experiments shown in this section is to analyze a multi-objective optimization for the provisioning of resources in clouds that considers the Pareto trade-off between cost and makespan by applying different service types for different SLAs. The experiments were carried out in the CloudSim Simulator 3.0.3 3 version[8], 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.

As previously mentioned, we used the SLA generator from [9]. Table 4 presents the SLAs applied to the validation with the QoS target, one SLA for each service type.

TABLE 4. SLAs generated.

Service	VMs	Makespan	Cost	Workload*
Smallpt	X*	125	3,12	90500
Apache	Y*	85	5,53	138000

*VMs it's the output of the algorithm.

NGSA-II and MPGA were executed 30 times for each SLA, which means that the same experiment was replicated for both algorithms. Table 5 shows the parameter settings for each algorithm. These values were empirically defined, based on previous settings reported in the literature ([31] [9]). The only parameter that differs between NSGA-II and MPGA is the number of individuals, since it is not possible to define a good frontier to NSGA-II with only 5 individuals.

TABLE 5. Algorithms settings

Characteristics	NSGA-II	MPGA
Time execution (minutes)	5	5
alpha	0.2	0.2
Operators	All	All
Individuals	50	5
Populations	-	3
Mutation rate	0.7	0.7
Crossover rate	0.5	5.0

Figure 9 shows the results for the NSGA-II and MPGA experiment using the Apache Benchmark. We have analysed the number of frontiers returned during the optimization process within 30 seconds of NSGA-II execution, and we keep the mapping of frontiers for the next 60, 90 and 120 seconds. After this time, the frontiers overlapped the frontier obtained in 120 seconds. Therefore, there is no improvement at the Pareto frontier over 120 seconds.

Figure 10 shows the frontiers found within 30, 60, 90 and 120 seconds. A total of 12 frontiers was found within 30 seconds for all 30 executions and 27 frontier within 60 seconds. There is a variation in the number of frontier after 90 seconds of execution.

Figures 11 and 12 show the analogue results for Smalpt benchmark. It's important to note that while the single objective method returned only one solution, a multi-criteria algorithm provides many other possibilities. Therefore, it's

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

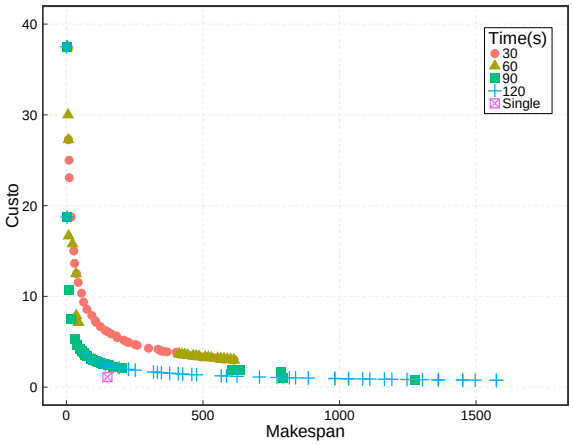

FIGURE 9. Pareto frontier solutions considering cost and makespan as an objective function for the Smallpt benchmark.

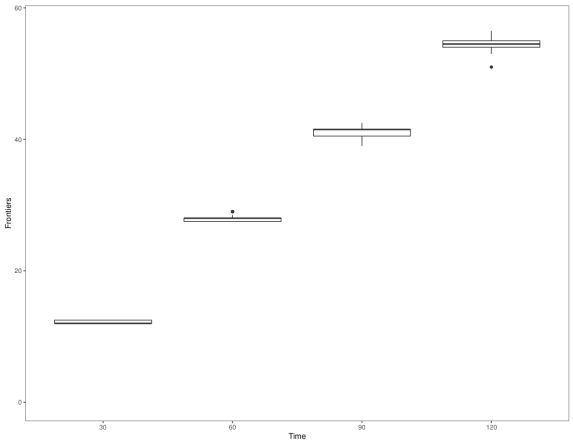

FIGURE 10. Number of Pareto frontiers found for different optimization times for the Smallpt benchmark.

possible to see that multi-criteria analysis is a more robust method than a single objective one. Furthermore, in section V, we have mentioned that the clients do not always have the technical knowledge to handle the provisioning of resources, for this reason, a multi-objective approach is essential to evidence that there are a lot of other possible choices.

Finally, Tables 6 and 7 compare the SLAs from the MPGA and NSGA-II solutions. The NSGA-II solutions are those ac hived in the frontier defined within 120 seconds of execution time, one where makespan is priority, one where cost is priority, and another one where the aim is the trade-off. It only presents one SLA for MPGA solution, as it is a single objective method.

TABLE 6. Solutions found for MPGA and NSGA-II by switching the priority of each parameter to the Apache benchmark.

Client	Priority	Capacity	Makespan	cost
3*Y	Makespan	0-1-18	15	35,7145
	Cost	0-1-0	728,0476	1,8382
	Trade Off	0-1-7	85,0363	7,417
	MPGA	0-1-4	166.66	2,8

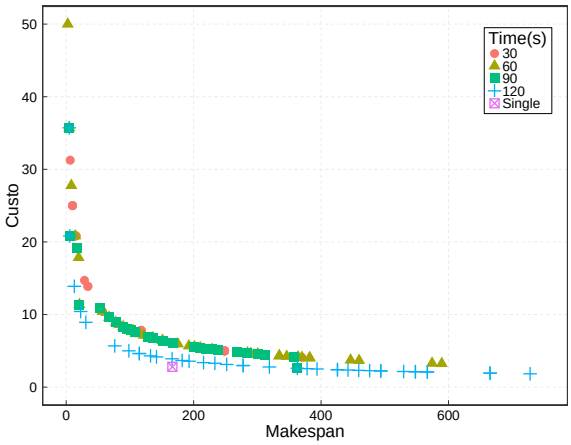

FIGURE 11. Pareto frontier solutions considering cost and makespan as an objective function for the Apache benchmark.

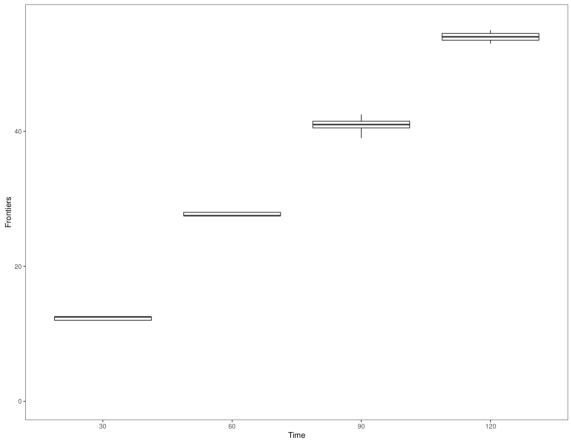

FIGURE 12. Number of Pareto frontiers found for different optimization times for the Apache benchmark.

TABLE 7. Solutions found for MPGA and NSGA-II by switching the priority of each parameter to the Smallpt benchmark

Client	Priority	Capacity	Makespan	cost
3*X	Makespan	0-14-0	37,8572	17,4848
	Cost	1-0-0	1574,4976	0,75
	Trade Off	0-3-3	110,71	2,417
	MPGA	0-0-3	150.0019	1.099

VII. CONCLUSIONS

It is not a trivial task to provide cloud computing clients with an efficient infrastructure, that respects their SLA and its QoS attributes, while at the same time seeking to reduce costs. Within the domain of cloud ecosystem, 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 article addresses one of these challenges, namely how to provide the client with an infrastructure that provides an SLA agreed between the client and provider in a multi-objective way. Our proposal mapped some of the QoS attributes that determine the criteria for the SLA and we also designed and analyzed algorithms that allow an optimized (re)configuration of the infrastructure based on these criteria. The results provide evidence that the NSGA-II algorithm is efficient and applicable to the solution of the problem, providing flexibility on the SLA.

In future work, we intend to carry out new tests with a prototype trather than simulator. We are also going to design other QoS attributes to be added to the SLA and apply the approach to other kinds of services. Furthermore, we intend to evaluate other multi-criteria methods to see whether further improvements can be made.

REFERENCES

[1] Apache benchmarking. Available from: https://openbenchmarking.org/test/pts/apache. Accessed: 06-27-2017.
[2] Smallpt benchmarking. Available from: https://openbenchmarking.org/test/pts/smallpt. Accessed: 06-27-2017.
[3] B. G. Batista, J. C. Estrella, C. H. G. Ferreira, D. M. Leite Filho, L. H. V. Nakamura, S. Reiff-Marganiec, M. J. Santana, and R. H. C. Santana. Performance evaluation of resource management in cloud computing environments. PloS one, 10(11):e0141914, 2015.
[4] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future generation computer systems, 28(5):755–768, 2012.
[5] I. Brandic, V. C. Emeakaroha, M. Maurer, S. Dustdar, S. Acs, A. Kertesz, and G. Kecskemeti. Laysi: A layered approach for sla-violation propagation in self-manageable cloud infrastructures. In Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual, pages 365–370. IEEE, 2010.
[6] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng. Cost optimized provisioning of elastic resources for application workflows. Future Generation Computer Systems, 27(8):1011–1026, 2011.
[7] H. E. Carvalho and O. C. Duarte. Voltaic: volume optimization layer to assign cloud resources. In Proceedings of the 3rd International Conference on Information and Communication Systems, page 3. ACM, 2012.
[8] Y. Chen, X. Li, and F. Chen. Overview and analysis of cloud computing research and application. In E-Business and E-Government (ICEE), 2011 International Conference on, pages 1–4. IEEE, 2011.
[9] L. J. M. de Azevedo, J. C. Estrella, L. H. V. Nakamura, M. J. Santana, R. H. C. Santana, C. F. M. Toledo, B. G. Batista, and S. Reiff-Marganiec. Optimized service level agreement establishment in cloud computing. The Computer Journal, pages 1–14, 2017.
[10] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE transactions on evolutionary computation, 6(2):182–197, 2002.
[11] A. Eiben and J. Smith. Introduction to Evolutionary Computing. Natural Computing Series. Springer Berlin Heidelberg, 2007.
[12] V. C. Emeakaroha, I. Brandic, M. Maurer, and I. Breskovic. Sla-aware application deployment and resource allocation in clouds. In Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual, pages 298–303. IEEE, 2011.
[13] L. Eyraud-Dubois and H. Larchevêque. Optimizing resource allocation while handling sla violations in cloud computing platforms. In Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, pages 79–87, 2013.
[14] F. Faniyi and R. Bahsoon. A systematic review of service level management in the cloud. ACM Computing Surveys (CSUR), 48(3):43, 2015.
[15] M. Guzek, P. Bouvry, and E.-G. Talbi. A survey of evolutionary computation for resource management of processing in cloud computing. Computational Intelligence Magazine, IEEE, 10(2):53–67, 2015.
[16] W. Ho, X. Xu, and P. K. Dey. Multi-criteria decision making approaches for supplier evaluation and selection: A literature review. European Journal of Operational Research, 202(1):16 – 24, 2010.
[17] T. T. Huu and J. Montagnat. Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure. In Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, pages 612–617, 2010.
[18] B. Jennings and R. Stadler. Resource management in clouds: Survey and research challenges. Journal of Network and Systems Management, 23(3):567–619, 2015.

[19] J. Jiang, J. Lu, G. Zhang, and G. Long. Optimal cloud resource auto-scaling for web applications. In Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on, pages 58–65. IEEE, 2013.
[20] S.-Y. Jing, S. Ali, K. She, and Y. Zhong. State-of-the-art research study for green cloud computing. The Journal of Supercomputing, 65(1):445–468, 2013.
[21] Y. Kessaci, N. Melab, and E.-G. Talbi. 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, 2014. Special Section: Intelligent Big Data ProcessingSpecial Section: Behavior Data Security Issues in Network Information PropagationSpecial Section: Energy-efficiency in Large Distributed Computing ArchitecturesSpecial Section: eScience Infrastructure and Applications.
[22] J. M. Ko, C. O. Kim, and I.-H. Kwon. Quality-of-service oriented web service composition algorithm and planning architecture. Journal of Systems and Software, 81(11):2079–2090, 2008.
[23] A. Kochut, Y. Deng, M. R. Head, J. Munson, A. Sailer, H. Shaikh, C. Tang, A. Amies, M. Beaton, D. Geiss, et al. Evolution of the ibm cloud: Enabling an enterprise cloud services ecosystem. IBM Journal of Research and Development, 55(6):7–1, 2011.
[24] G. Kousiouris, T. Cucinotta, and T. Varvarigou. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. Journal of Systems and Software, 84(8):1270–1291, 2011.
[25] A. Kumbhare, Y. Simmhan, and V. K. Prasanna. Exploiting application dynamism and cloud elasticity for continuous dataflows. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, page 57. ACM, 2013.
[26] A. G. Kumbhare, Y. Simmhan, M. Frincu, and V. K. Prasanna. Reactive resource provisioning heuristics for dynamic dataflows on cloud infrastructure. Cloud Computing, IEEE Transactions on, 3(2):105–118, 2015.
[27] D. B. L.D. and P. V. Krishna. Honey bee behavior inspired load balancing of tasks in cloud computing environments. Applied Soft Computing, 13(5):2292 – 2303, 2013.
[28] M. Masdari, S. S. Nabavi, and V. Ahmadi. An overview of virtual machine placement schemes in cloud computing. Journal of Network and Computer Applications, 66:106–127, 2016.
[29] J. W. Rittinghouse and J. F. Ransome. Cloud computing: implementation, management, and security. CRC press, 2016.
[30] G. Soni and M. Kalra. A novel approach for load balancing in cloud data center. In Advance Computing Conference (IACC), 2014 IEEE International, pages 807–812. IEEE, 2014.
[31] C. F. M. Toledo, M. da Silva Arantes, R. R. R. De Oliveira, and B. Almada-Lobo. Glass container production scheduling through hybrid multi-population based evolutionary algorithm. Applied Soft Computing, 13(3):1352–1364, 2013.
[32] X. Wang, Z. Du, and Y. Chen. An adaptive model-free resource and power management approach for multi-tier cloud environments. Journal of Systems and Software, 85(5):1135–1146, 2012.
[33] G. J. Woeginger. Exact algorithms for np-hard problems: A survey. In Combinatorial Optimization—Eureka, You Shrink!, pages 185–207. Springer, 2003.
[34] L. Wu, S. K. Garg, and R. Buyya. Sla-based admission control for a software-as-a-service provider in cloud computing environments. Journal of Computer and System Sciences, 78(5):1280–1299, 2012.
[35] Q. Yan, F. R. Yu, Q. Gong, and J. Li. Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. IEEE Communications Surveys & Tutorials, 18(1):602–622, 2016.
[36] Y. O. Yazir, Y. Akbulut, R. Farahbod, A. Guitouni, S. W. Neville, S. Ganti, and Y. Coady. Autonomous resource consolidation management in clouds using impromptu extensions. In Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, pages 614–621, 2012.
[37] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications, 1(1):7–18, 2010.

LEONILDO JOSÉ DE MELO DE AZEVEDO is currently a Ph.D. candidate in Mathematical and Computer Sciences in the Institute of Mathematical and Computer Sciences (ICMC) at the University of São Paulo (USP). Leonildo holds a Master of Science degree from the University of São Paulo, and a degree in Computer Science from the Midwestern State University (UNICENTRO). His areas of interest in research include Distributed Computing, Cloud Computing, Optimization, Metaheuristics, and Evolutionary Algorithms.

JÚLIO CEZAR ESTRELLA is an associate professor at the Institute of Mathematics and Computer Science (ICMC - USP); received a bachelor's degree in Computer Science at the State University of São Paulo - Julio de Mesquita Filho (UNESP) in 2002 and a Ph.D. in Computer Science from the University of São Paulo (USP) in 2010. He has been a researcher at ICMC-USP since 2011. In 2016 he became an associate researcher at ICMC-USP. Julio is an expert in Computer Systems Architecture and works on dynamic resources provisioning applied to Cloud Computing, Internet of Things (IoT), with ramifications for Smart Cities, Service-Oriented Architectures(SOA), Computer Networks and Computer Security.

CLAUDIO FABIANO MOTTA TOLEDO holds a degree in Applied and Computational Mathematics from the State University of Campinas (1995), a Master in Electrical Engineering from the State University of Campinas (1999), and a Ph.D. in Electrical Engineering from the State University of Campinas (2005). He is currently an associate professor at the University of São Paulo (USP). Claudio has experience in Computer Science, focusing on Evolutionary Systems, working on the following topics: production scheduling, optimization, batch sizing, metaheuristics, and evolutionary algorithms.

STEPHAN REIFF-MARGANIEC is a Professor of Computer Science and Head of School of Electronics, Computing and Maths at the University of Derby (UK). He worked in the computer industry in Germany and Luxembourg, holding research positions at the University of Glasgow and Stirling University (while pursuing a Ph.D.). Stephan lead work in the European Union (EU) on projects funded by Leg2Net, Sensoria, and InContext focusing on automatic service adaptation, context-aware service selection, workflow-based service composition, and rules. Stephan is co-editor of the textit Handbook of Research on Service-Oriented Systems and Non-Functional Properties and has published over 100 articles in international conferences and magazines, and has served on a large number of program committees. Stephan was appointed visiting professor at the China Petroleum University and visiting professor at Lamsade at Dauphine University in Paris. He is a member of ACM and IEEE.