Hindawi
Scientific Programming
Volume 2020, Article ID 4653204, 17 pages
https://doi.org/10.1155/2020/4653204

Research Article

A Comparative Analysis of NSGA-II and NSGA-III for Autoscaling Parameter Sweep Experiments in the Cloud

Virginia Yannibelli,[1] Elina Pacini,[2,3] David Monge,[4] Cristian Mateos,[1] and Guillermo Rodriguez[1]

[1]*ISISTAN (UNICEN-CONICET), Tandil, Buenos Aires, Argentina*
[2]*Facultad de Ingeniería, Universidad Nacional de Cuyo, Mendoza, Argentina*
[3]*Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Buenos Aires, Argentina*
[4]*ITIC, UNCUYO, Mendoza, Argentina*

Correspondence should be addressed to Guillermo Rodriguez; exarodriguez@gmail.com

Received 19 December 2019; Revised 27 June 2020; Accepted 20 July 2020; Published 28 August 2020

Academic Editor: Basilio B. Fraguela

The Cloud Computing paradigm is focused on the provisioning of reliable and scalable virtual infrastructures that deliver execution and storage services. This paradigm is particularly suitable to solve resource-greedy scientific computing applications such as parameter sweep experiments (PSEs). Through the implementation of autoscalers, the virtual infrastructure can be scaled up and down by acquiring or terminating instances of virtual machines (VMs) at the time that application tasks are being scheduled. In this paper, we extend an existing study centered in a state-of-the-art autoscaler called multiobjective evolutionary autoscaler (MOEA). MOEA uses a multiobjective optimization algorithm to determine the set of possible virtual infrastructure settings. In this context, the performance of MOEA is greatly influenced by the underlying optimization algorithm used and its tuning. Therefore, we analyze two well-known multiobjective evolutionary algorithms (NSGA-II and NSGA-III) and how they impact on the performance of the MOEA autoscaler. Simulated experiments with three real-world PSEs show that MOEA gets significantly improved when using NSGA-III instead of NSGA-II due to the former provides a better exploitation versus exploration trade-off.

1. Introduction

Cloud Computing [1] dynamically provides on-demand and scalable resources to support scientific application execution. The existence of Clouds allows scientists to facilitate the execution of large-scale computational experiments by exploiting existing hardware resources. A representative example of these experiments is parameter sweep experiments (PSEs) [2, 3]. An example of PSE is the one discussed in [2] where a plane strain plate with a central circular hole under imposed displacements stretching the plate was studied. In the study, the authors considered different viscosity values and other constitutive model parameters to adjust the model response. Then, through the variation of the parameters, a lot of study cases were obtained, which was necessary to run in parallel. Therefore, to achieve an efficient execution of the PSE tasks, infrastructures such as Cloud are necessary.

A Cloud enables for the acquisition of computing resources through different types of virtual machine (VM) instances [4] that are provided with a wide spectrum of hardware and software configurations under a pay-per-use scheme. Moreover, VM prices differ according to the type of instance acquired, where the type is in turn a combination of the number of available virtual CPUs and maximum RAM/disk space that can be allocated. In addition, instance prices may also vary according to the pricing model of the Cloud provider. First, there are on-demand VMs that can be accessed for a fixed price. The price is usually charged by the hour of use. Second, the spot instances have fluctuating prices over time. The prices tend to decrease during low-demand periods. On-demand instances are more expensive

than spots, but spot instances are subject to abrupt terminations by the provider. The user, for acquiring a spot instance, must bid the maximum price that he/she is willing to pay. Then, if the spot price overcomes such bid, an out-of-bid (OOB) error occurs and the executing VM instances are terminated after a short period. Moreover, it is important to mention that, when the instances are unexpectedly finished, this negatively impacts the task execution times, which causes delays in the whole PSE.

Since the workloads in a Cloud are very often caused by the coexistence of applications with different computing requirements, autoscaling strategies [5–7] become necessary. Cloud autoscaling is a strategy that is responsible, on the one hand, for determining the appropriate number/type of instances to use and, on the other hand, for scheduling in an online fashion the PSE tasks on the running infrastructure represented by the acquired instances. These are two interdependent problems that must be solved simultaneously. It is important to mention that since the applications usually comprise a large number of tasks and that a wide spectrum of VM instance types and pricing models are available, determining beforehand the right amount and type of necessary instances for an application is not a trivial problem. For solving this problem, it is necessary to consider the dynamic resource availability variations on the virtual infrastructure.

Concretely, in this work, we address the autoscaling problem for PSEs, to determine the right number of each VM instance type and pricing model as well as the bid prices for the spot instances. Specifically, we rely on a formulation of our problem that involves minimizing the makespan, i.e., the total execution time of all tasks in a PSE, while also minimizing the monetary cost of this set of PSE tasks and the OOB errors, while focusing on the algorithm that best optimizes the formulated problem.

From the related works, most Cloud autoscaling strategies have been proposed for executing workflow applications and are mainly subject to deadline constraints [8–10]. However, these approaches are not appropriate for managing PSEs because workflows are applications that have different computational requirements. The main difference is that in a PSE, tasks are independent (i.e., there are no intertask dependencies) and somewhat homogeneous [3], which determines a completely different autoscaling scenario. Besides, from the surveyed works, only in the work in [11], the authors have addressed a multiobjective minimization problem aiming at reducing the makespan, the monetary cost, and the potential impact of OOB errors for different PSEs. Such autoscaler is called multiobjective evolutionary autoscaler (MOEA) and is based on the multiobjective evolutionary algorithm called nondominated sorting genetic algorithm II (NSGA-II). The key idea of this work is to propose a variant of the MOEA autoscaler based on NSGA-III to solve the autoscaling of PSEs for scenarios where the availability of the virtual infrastructure varies constantly throughout the applications execution. Therefore, this article brings in the following contributions:

(i) A new and improved multiobjective Cloud autoscaler of the autoscaler proposed in [11] for executing PSEs: the new autoscaler is based on the nondominated sorting genetic algorithm III (NSGA-III). Specifically, the key idea of this article is to provide a new and improved Cloud autoscaler for PSEs capable of determining the right amount of VM instances for each VM type for minimizing the makespan, the monetary cost, and the impact of OOB errors. As far as we know, autoscalers based on the NSGA-III have not been proposed in the literature. Besides, this algorithm has demonstrated to be very efficient with respect to other known similar algorithms, such as NSGA-II, for solving various NP-hard problems [12–14]. The mathematical formulation of the problem and the MOEA autoscaler are described in Sections 2 and 3, respectively. Then, Section 4 describes the NSGA-II and NSGA-III algorithms.

(ii) The proper parameterization values for the NSGA-II and NSGA-III: we present an experimental validation demonstrating that by using NSGA-III, the performance of the MOEA autoscaler can be greatly boosted considering the results already obtained in [1], w.r.t. relevant metrics, particularly makespan, economic cost, number of tasks failures derived from OOB errors, and the L_2-norm [15] of such metrics. The experiments were performed considering nine study cases, derived from two real-world PSEs, involving a viscoplastic and an elastoplastic problem from the computational mechanics domain, and an ensemble of the two PSEs, over simulated Cloud environments [16]. Such applications and the actual characteristics of the instances used and their prices, extracted from the popular Amazon EC2 Cloud service, are described in Section 5. Section 6 discusses the experiments and presents the results, while also discussing the statistical significance tests that back up the strength of our claims.

Finally, Section 7 surveys relevant related work focused on addressing the autoscaling problem, and Section 8 concludes this work.

2. Multiobjective Autoscaling Problem

In this paper, we address the problem of autoscaling PSEs in public Clouds (e.g., Amazon EC2 and Google Cloud) with the help of evolutionary algorithms. This general problem involves determining the number and type of VM instances to be acquired to execute the tasks in a given PSE, and scheduling these tasks on the instances, so that the predefined optimization objectives are reached. In this respect, we consider three relevant optimization objectives: the minimization of the makespan, the minimization of the monetary cost, and the minimization of the impact of out-of-bid (OOB) errors. This problem is recognized as a multiobjective NP-hard problem [11].

A PSE model is a numerical experiment which involves a large number of tasks to be executed and usually needs several hours or days to be completed. In practice, when a PSE is executed, the execution, duration, and cost of the tasks differ according to the instances used to execute these tasks.

In public Clouds, instances of different types of VMs have different characteristics with respect to operating system, number of processors, processing power, memory size, memory speed, disk size, and the like. Moreover, instances of different types of VMs have different monetary cost. In this respect, instances of VMs can be acquired under different pricing models, which determine the cost of instances and also their behavior.

In this paper, we consider that instances are charged for every hour used under two possible pricing models, namely, the on-demand model and the spots model, as in real public Clouds (e.g., Amazon EC2). In the on-demand model, instances are acquired at a fixed monetary cost by the hour of computation. The monetary cost of instances acquired under this pricing model is usually higher than that of instances acquired under the spots model. For the remainder of this work, we will refer to instances acquired under the on-demand model as on-demand instances.

In the spots pricing model, the monetary cost of instances fluctuates mostly unpredictably over time and is usually lower than that of the on-demand instances. To acquire an instance under this pricing model, the user must make a bid as in a stock market, indicating the maximum amount of money he/she is willing to pay for the instance. Then, while the current monetary cost of the instance is lower than the bid of the user, the instance will remain assigned to the user. Otherwise, when the current monetary cost of the instance exceeds the bid of the user, an OOB error happens. This error forces the termination of the instance, and therefore, the abrupt termination of the execution of the tasks running on the instance. Thus, OOB errors can impact negatively on the execution of PSEs. For the remainder of this work, we will refer to the instances acquired under the spots model as spots instances.

The PSE autoscaling problem addressed here is composed of two well-defined interrelated problems. The first problem involves determining a scaling plan detailing the number and type of on-demand and spot instances to request to the Cloud provider for the next hour and also the bids corresponding to the spot instances (i.e., the virtual infrastructure setting to request for the following hour). The second problem involves scheduling the tasks of the PSE on both the on-demand and spot instances acquired.

These two problems need to be solved periodically (i.e., every one hour) during the execution of the PSE, in order to update the infrastructure setting according to the workload of the PSE (i.e., the tasks whose execution is pending) and schedule this workload on the new infrastructure setting, so that the considered optimization objectives are reached. It is important to note that the autoscaler reevaluation conditions are not the same in each autoscaling cycle. For example, the price of spot instances varies because it is subject to a supply and demand market, the Cloud provider may suddenly finish some of the acquired spot instances, and therefore, less computing power will be available, and finally, it is possible that the set of running tasks and ready-to-run tasks change (in each cycle, there may be tasks that have already been completed and/or tasks that have been canceled). In Section 2.1, we present the mathematical formulation of the addressed PSE autoscaling problem, which sets the formal basis for these two interrelated problems.

2.1. Mathematical Formulation of the Problem. This section describes the mathematical formulation of the problem [11], in order to make the paper self-contained. Given I, the set of instance types considered for autoscaling, and $n = |I|$, the number of instance types, a scaling plan X is formally defined as a 3-tuple (x^{od}, x^s, and x^b), where

(i) $x^{od} = (x_1^{od}, x_2^{od}, \ldots, x_n^{od})$ is a vector which describes the number of on-demand instances to be acquired for each of the n instance types. Here, each x_i^{od} is an integer value that ranges between 0 and the maximum number of on-demand instances that establishes the Cloud provider regarding the type i for the current autoscaling stage (i.e., for the next hour).

(ii) $x^s = (x_1^s, x_2^s, \ldots, x_n^s)$ is a vector which describes the number of spot instances to be acquired for each of the n instance types. Here, each x_i^s is an integer value that ranges between 0 and the maximum number of spot instances that establishes the Cloud provider regarding the type i for the current autoscaling stage.

(iii) $x^b = (x_1^b, x_2^b, \ldots, x_n^b)$ is a vector which describes the bid price for the spot instances of each of the n instance types. Here, each x_i^b is a real value that ranges between the current spot price and the on-demand price for the instance type i.

Given the set T of PSE tasks considered for the current autoscaling stage (i.e., the tasks whose execution is pending, including tasks that started in previous stages and are running yet), the multiobjective autoscaling problem inherent to this autoscaling stage is defined as the minimization of the three objective functions shown in equation (1), subject to a set of constraints. This is the step we focus on this paper, as this is tackled by a genetic algorithm:

$$\min(\text{makespan}(X), \text{cost}(X), \text{errorsImpact}(X)). \quad (1)$$

2.1.1. Objective Functions. The three objective functions considered as part of the problem are described below.

(1) Makespan. Equation (2) presents the definition of the makespan $(\cdot)$ function. Makespan is computed as the execution time of the set T of tasks. For computing makespan, the execution of tasks is simulated by scheduling the tasks in T on the set of instances described by x^{od} and x^s. Tasks are greedily scheduled using the earliest completion time (ECT)

criterion, i.e., scheduling each task to the instance that promises the earliest completion time.

Formally, the makespan is computed as

$$\text{makespan}(X) = \max_{t \in T}\{ST(t) + d_t\} - \min_{t \in T}\{ST(t)\}, \quad (2)$$

where d_t is the duration of the task t. Here, such duration corresponds to the expected execution time of the task t on the instance chosen with the ECT criterion. $ST(t)$ is the start time of the task t according to the schedule generated by the ECT algorithm.

(2) Cost. Equation (3) presents the definition of the cost $(\cdot)$ function, which represents the cost of running the instances indicated in x^{od} and x^s for one hour of computation:

$$\text{cost}(X) = \sum_{i=1}^{n} x_i^{od} \times \text{price}_i + x_i^s \times x_i^b. \quad (3)$$

The cost of on-demand instances is the sum of running all the on-demand instances of each type i (x_i^{od}) multiplied by their corresponding on-demand price (price_i). The cost of spot instances is computed differently. The model computes such cost as the product of the number of spot instances of type i (x_i^s) and their corresponding bid (x_i^b).

It is worth to point out that equation (3) computes a pessimistic estimation of the overall cost because of two reasons. First, the equation does not consider the occurrence of OOB errors, therefore assuming that the instance is charged for the whole interval. Second, by using the bid as a proxy of the spot price, upper bounds of the actual spot prices are considered.

(3) Potential Impact of OOB Errors. Equation (4) presents the definition of the errorsImpact $(\cdot)$, function which measures the potential impact of OOB errors on the execution of the PSE tasks according to the number of spot instances and their bids. As the occurrence of OOB errors depends on the bid and evolution of spot prices, it is not possible to determine beforehand if OOB errors will occur or when they will occur. For such a reason, we use a function that models the probability of OOB-error occurrences for different bids. Formally, the impact of OOB errors on the tasks is computed as

$$\text{errors Impact}(X) = \sum_{i=1}^{n} x_i^s \times \nu\,\text{CPU}_i \times P_i\left(x_i^b\right). \quad (4)$$

The potential impact of OOB errors is computed as the sum of the total number of virtual CPUs of type i ($x_i^s \times \nu\text{CPU}_i$) weighted by the probability of OOB error given the bid for such type ($P_i(x_i^b)$). The probability function $P_i(\cdot)$ for the instance of type i can be easily computed offline by counting the number of OOB occurrences for a number of predefined bid levels over a history of spot prices. The probability function is computed as

$$P_i\left(x_i^b\right) = \frac{1}{w}\left|\left\{S_{ij} \;\middle|\; \exists s_{ij}^{(k)} \in S_{ij},\ s_{ij}^{(k)} > x_i^b\right\}\right|,\ 1 \le k \le m \quad (5)$$

where $S_{ij} = \left\{s_{ij}^{(1)}, \ldots, s_{ij}^{(m)}\right\}$ is the j^{th} function of w subseries of spot prices obtained from the available historical data for the VM type i (in Section 5.2, we provide details about the historical data considered). Then, the formula computes the average number of series for which at least one value is greater than the bid price x_i^b. These subseries of spot prices have duration of 1 hour, which is the minimum charge unit of any VM. Note that the bid price x_i^b is constrained to take the same values described as part of the autoscaling problem definition. In this study, these are 10 values for each VM type.

The image of the errorsImpact $(\cdot)$ function is $[0, \sum_{i=1}^{n} x_i^s \times \nu\,\text{CPU}_i]$, where 0 corresponds to the cases in which there are no tasks affected by OOB errors (i.e., there are no task failures), and $\sum_{i=1}^{n} x_i^s \times \nu\,\text{CPU}_i$ corresponds to the cases in which all the tasks have a probability of 1 of being affected.

2.1.2. Constraints. The set of constraints considered as part of the problem is described below.

(1) Budget Constraint. Equation (6) presents the budget constraint which limits the cost of the scaling plan to be below the maximum monetary budget B:

$$\text{cost}(X) \le B. \quad (6)$$

(2) Instance Constraints. Equation (7) presents the set of n constraints that set the minimum and the maximum number of instances for each type. In this equation, $X_i^{\min}$ and $X_i^{\max}$ are, respectively, the minimum and maximum number of allowed instances regarding the instance type i for the current autoscaling stage. The minimum amount of instances ($X_i^{\min}$) refers to the amount of running instances of type i that are currently processing at least one task:

$$X_i^{\min} \le x_i^{od} + x_i^s \le X_i^{\max}. \quad (7)$$

The constraint in equation (8) forces that the scaling plan must contain at least one instance:

$$\sum_{i=1}^{n} x_i^{od} + x_i^s \ge 1. \quad (8)$$

(3) Bid Constraints. Equation (9) presents the set of n constraints aimed at circumscribing bids to be between the current spot price and a maximum spot price allowed. In this equation, $S_i^{\text{current Price}}$ is the current spot price for the instances of type i, and price_i is the on-demand price for the instances of type i:

$$S_i^{\text{current Price}} \le x_i^b \le \text{price}_i. \quad (9)$$

3. The MOEA Autoscaler

As described in Section 2, throughout the execution of a PSE, different autoscaling stages need be addressed periodically (i.e., every one hour), which means that the multiobjective

autoscaling problems inherent to these different stages need be solved.

To solve each of these multiobjective autoscaling problems, the MOEA autoscaler [11] develops three sequential steps. In the first step, the algorithm NSGA-II [17] is used in order to obtain an approximation to the optimal Pareto set. In the second step, the best solution of the Pareto set obtained by the first step is selected. In the third step, the scaling plan represented by the selected solution is applied, and the scheduling of the tasks is developed.

3.1. First Step. This step is concerned with solving the multiobjective autoscaling problem described in Section 2.1.

In the MOEA autoscaler, this step is developed by applying the NSGA-II and, therefore, obtaining a Pareto set (i.e., set of nondominated solutions) where each solution represents a feasible scaling plan and is encoded like the 3-tuple (x^{od}, x^{s}, and x^{b}) detailed in Section 2.1.

In the study performed in this paper, this step is carried out by using the two optimization algorithms object of our analysis, i.e., NSGA-II as used in MOEA and NSGA-III [18]. In Section 4, we present the main characteristics of these algorithms and also the differences between them.

3.2. Second Step. From the Pareto set provided by the first step, one solution is selected to solve the addressed multiobjective autoscaling problem, by applying a predetermined selection criterion. This is meant in order to obtain one solution in a fully autonomous way, without relying on a human decision maker.

The selection criterion calculates the distance of each solution of the Pareto set to an ideal solution, by applying the well-known L_2-norm [15]. Then, the criterion selects the solution which minimizes the distance to the ideal solution. In this case, the ideal solution corresponds to the solution with makespan, cost, and task failure probability equal to 0. Thus, the criterion considers the trade-off of each solution of the Pareto set among the optimization objectives of the addressed problem. Note that the makespan, cost, and task failure probability of each solution of the Pareto set are calculated by equations (2)–(4), respectively.

3.3. Third Step. From the solution selected by the second step, the autoscaler builds the scaling plan. Specifically, the autoscaler acquires the number of on-demand instances detailed in the solution and the number of spot instances with the corresponding bid prices detailed in the solution. Then, the autoscaler schedules the tasks in T on the on-demand and spot instances acquired, by applying the ECT criterion. Note that T refers to the set of PSE tasks considered for the current autoscaling stage, as mentioned in Section 2.1.

4. Optimization Algorithms

This section describes the multiobjective optimization algorithms studied to solve the multiobjective autoscaling problem that takes place in the first step of the execution of the MOEA autoscaler.

4.1. NSGA-II. NSGA-II [17] is a known multiobjective evolutionary algorithm which has been widely applied in the literature [15]. This algorithm begins creating a random initial population with s solutions, where s is a given number. In this case, each solution represents a feasible scaling plan and is encoded as detailed in Section 4.3.

In each generation, the algorithm selects s solutions from the current population to compose the mating pool, by applying the traditional binary tournament selection process. This process considers the nondomination level of the solutions which is defined by applying the nondominated sorting [15] to the current population. The process also considers the crowding distance of the solutions which represents the distance between a solution and its neighbors. Then, the process considers the following criterion to determine the best solution of each of the s tournaments. When a tournament involves two solutions with different nondomination levels, the solution with the lower (better) level is preferred and selected for the mating pool. Otherwise, when the two solutions of a tournament have the same nondomination level, the solution with the higher crowding distance is preferred and selected, in order to promote the selection of diverse nondominated solutions for the mating pool.

After the mating pool is composed, the algorithm creates an offspring population from the mating pool, by applying the known simulated binary crossover operator [15] and then the known polynomial mutation operator [15]. The crossover operator is applied with a crossover probability Pc and a crossover distribution index Dc, and the mutation operator is applied with a mutation probability Pm and a mutation distribution index Dm.

Once the offspring population is created, the algorithm combines the current and offspring populations and then selects the best s solutions from this combined population, to create a new population for the next generation. To select these s solutions, the algorithm applies the nondominated sorting [15] to the combined population. After that, the solutions in this population are grouped according to their nondomination levels as $\{F_1, F_2, \ldots\}$. Then, each nondomination level is selected one at a time to create the new population, beginning from F_1, until the size of the new population is equal to s or higher than s. When the size of the new population is equal to s, additional selection operations are not needed, and the next generation starts from this population. When the size of the new population is higher than s, the last nondomination level selected F_l is not fully included in this population. In this respect, the solutions from level F_1 to level F_{l-1} are included in this population, and the r remaining solutions ($r = s - |F_1 \cup \ldots \cup F_{l-1}|$) are selected from level F_l.

To select the r remaining solutions from the level F_l, the algorithm utilizes a selection process which considers the crowding distances of the solutions in F_l. This process

calculates the crowding distance of each solution in F_l. Then, the process sorts the solutions in F_l based on their distances in descending order and selects the r solutions with the highest distances. Thus, this process aims to promote the selection of diverse nondominated solutions, with the aim of preserving the diversity of the new population.

The algorithm ends its execution once a predefined termination criterion is achieved (i.e., a given number of evaluations for the generated solutions), providing the Pareto set (i.e., the set of nondominated solutions) of the population corresponding to the last generation.

4.2. NSGA-III. NSGA-III [18] is an extension of the framework of NSGA-II, which has been proposed with the aim of improving the performance of NSGA-II (i.e, the Pareto sets provided by NSGA-II) for many objective problems (i.e., problems with 3 or more objectives).

NSGA-III has the same general behavior as NSGA-II and uses the same procedures used by NSGA-II to develop some stages of the evolutionary cycle. In this respect, NSGA-III applies the same procedure used by NSGA-II to create the initial population with s solutions. In this case, each solution in this population represents a feasible scaling plan and is encoded as detailed in Section 4.3. Then, NSGA-III applies the same crossover and mutation operators applied by NSGA-II, to create an offspring population from the current population.

However, NSGA-III differs from NSGA-II in terms of the selection process utilized to create a new population with s solutions for the next generation, from the combined current and offspring population. The process utilized by NSGA-III starts as that utilized by NSGA-II, grouping to the solutions in the combined population according to their nondomination levels as $\{F_1, F_2, \ldots\}$ and then selecting the nondomination levels once at a time to build the new population, until the size of this population is greater or equal than s. However, when the size of the new population exceeds s and thus the last nondomination level selected F_l cannot be fully included in this population, NSGA-III uses a different selection process to that used by NSGA-II to decide which r solutions from F_l will be included in this population.

To select the remaining r solutions from level F_l, NSGA-III applies a selection process based on reference points. The process considers a set of reference points widely and uniformly distributed on the normalized hyperplane inherent to the optimization objectives of the problem addressed by the algorithm. Then, the process emphasizes the selection of solutions from F_l which are associated with each of these reference points. Thus, this process promotes the selection of diverse and well-distributed nondominated solutions, with the aim of preserving the diversity and distribution of the new population.

NSGA-III considers the same termination criterion used by NSGA-II to finish its execution. After such criterion is achieved, NSGA-III provides the Pareto set of the population corresponding to the last generation.

4.2.1. Determination of Reference Points. The algorithm NSGA-III utilizes the known Das and Dennis [19] systematic approach to determine the set of reference points to be used in each generation, as described exhaustively in [18].

This approach defines reference points on a normalized hyperplane that is equally inclined to all objective axes and has an intercept of one on each objective axis. The total number of reference points R in problems with M objectives is calculated by equation (10), where d is a given integer value which refers to the number of divisions considered along each objective axis. For example, in problems with three objectives ($M = 3$), the reference points are defined on a triangle with the apex at (1, 0, 0), (0, 1, 0), and (0, 0, 1). If twelve divisions ($d = 12$) are considered for each objective axis, 91 reference points will be created, according to equation (10). Thus, the reference points created by this approach are widely and uniformly distributed on the entire normalized hyperplane:

$$R = \binom{M + d - 1}{d}. \tag{10}$$

4.3. Encoding of Solutions. In the algorithms NSGA-II and NSGA-III studied, each solution of the population represents a feasible scaling plan for the current autoscaling stage.

Each solution is encoded as a vector with a size equal to $3 \times n$, where n is the number of instance types (i.e., the number of VM types) considered for autoscaling, as was mentioned in Section 2.1. In this vector, the positions $(1, n)$ indicate the number of on-demand instances to be acquired for each of the n instance types. These positions contain integer values that range between 0 and the maximum number of available on-demand instances for each of the n instance types. Then, the positions $(n + 1, 2 \times n)$ indicate the number of spot instances to be acquired for each of the n instance types. These positions contain integer values that range between 0 and the maximum number of available spot instances for each of the n instance types. Finally, the positions $((2 \times n) + 1, 3 \times n)$ indicate the bid price for the spot instances of each of the n instance types. These positions contain real values that range between the current spot price and the on-demand price for each of the n instance types. Note that this vector is like the 3-tuple (x^{od}, x^s, and x^b) presented in Section 2.1 to formally define a scaling plan.

4.4. Crossover Operator. The algorithms NSGA-II and NSGA-III apply the known simulated binary crossover operator [15] to generate offspring encoded solutions from pairs of parent encoded solutions in the mating pool. The operator is applied on each pair with a crossover probability Pc and a crossover distribution index Dc.

To generate the offspring solutions p_1' and p_2' from a given pair of parent solutions p_1 and p_2, the operator uses equations (11) and (12). In these equations, p_{1i}' and p_{2i}' refer to the values for the position i of the solutions p_1' and p_2', where $i = 1, \ldots, n$. The terms p_{1i} and p_{2i} refer to the values of

the position i of the solutions p_1 and p_2. The term B_i is calculated by equation (13), using polynomial probability distribution. In this equation, u_i is a random real number generated between 0 and 1. Dc is a given nonnegative real number. A large value of Dc provides a higher probability for generating offspring solutions near to parent solutions, allowing a focused search. A small value of Dc generates offspring solutions distant from parent solutions, allowing a diverse search:

$$p'_{1i} = 0.5\left[\left(1 + B_i\right) p_{1i} + \left(1 - B_i\right) p_{2i}\right], \tag{11}$$

$$p'_{2i} = 0.5\left[\left(1 - B_i\right) p_{1i} + \left(1 + B_i\right) p_{2i}\right], \tag{12}$$

$$B_i = \begin{cases} \left(2u_i\right)^{(1/(Dc+1))}, & \text{if } u_i \leq 0.5, \\ \left(\dfrac{1}{2\left(1 - u_i\right)}\right)^{(1/(Dc+1))}, & \text{if } u_i > 0.5. \end{cases} \tag{13}$$

4.5. Mutation Operator. The algorithms NSGA-II and NSGA-III apply the known polynomial mutation operator [15] on the encoded solutions generated by the crossover operator. This mutation operator is applied with a mutation probability Pm and a mutation distribution index Dm, on each position i of the encoded solutions, where $i = 1, \ldots, n$.

To generate a mutated value p'_i for the position i of a given encoded solution p, the operator uses equation (14). In this equation, u_i is a random real number generated between 0 and 1. The term p_i refers to the value of the position i of the solution p. The terms L_i and U_i correspond to the lower and upper bounds of the position i of the solution p, respectively. These terms are used to guarantee that no value outside the range (L_i and U_i) is created by the mutation operator. The term d is calculated by equation (15), using polynomial probability distribution. In this equation, the value of u_i corresponds to that used in equation (14). Dm is a given nonnegative real number:

$$p'_i = \begin{cases} p_i + d\left(p_i - L_i\right), & \text{if } u_i \leq 0.5, \\ p_i + d\left(U_i - p_i\right), & \text{if } u_i > 0.5, \end{cases} \tag{14}$$

$$d = \begin{cases} \left(2u_i\right)^{(1/(Dm+1))} - 1, & \text{if } u_i \leq 0.5, \\ 1 - \left(2\left(1 - u_i\right)\right)^{(1/(Dm+1))}, & \text{if } u_i > 0.5. \end{cases} \tag{15}$$

4.6. Differences between NSGA-II and NSGA-III. As was mentioned in Section 4.2, the algorithms NSGA-II and NSGA-III have a similar general behavior. Both algorithms start from a random initial population, where each solution represents a feasible scaling plan. In each generation, these algorithms use the same crossover and mutation operators to generate an offspring population from the current population. Nevertheless, these algorithms differ in relation to the selection process utilized to determine which solutions from the combined current and offspring population will compose the new population for the next generation.

In the NSGA-II, the selection process considers first the nondomination level of the solutions in the combined population and then the crowding distance of these solutions. The crowding distance represents the distance of a solution to its neighboring solutions. Then, the process emphasizes the selection of nondominated solutions with larger crowding distances. Thus, the process promotes the selection of diverse nondominated solutions. However, this process does not guarantee the selection of well-distributed nondominated solutions.

In contrast to NSGA-II, in NSGA-III, the selection process considers first the nondomination level of the solutions in the combined population and then the association of these solutions to the reference points. In this sense, the process utilizes a set of well-spread reference points (i.e., a set of widely and uniformly distributed reference points). Then, the process emphasizes the selection of nondominated solutions which are associated with each of these reference points. Thus, this process promotes the selection of diverse and well-distributed nondominated solutions, in order to preserve the diversity and distribution of the new population.

By using this selection process based on well-spread reference points, the algorithm NSGA-III has the possibility of reaching better Pareto sets in terms of both diversity and distribution of the nondominated solutions, these solutions represent feasible scaling plans for the multiobjective autoscaling problem addressed in the first step of the autoscaler. This represents the rationale of applying NSGA-III to the problem at hand. From now on, we empirically evaluate this claim.

5. Study Cases

In this Section, we detail the experimental setting used to evaluate the autoscaler based on each of the studied multiobjective genetic algorithms, i.e., NSGA-II and NSGA-III. First, Section 5.1 describes the three PSEs employed and how we derived from these PSEs the tasks to feed the CloudSim simulator to run the experiments. Then, Section 5.2 shows the characteristics of the VM instances used.

5.1. PSE Applications. PSEs are a very popular way of conducting simulation-based experiments, used by scientists and engineers, through which the same application code is run several times with different input parameters resulting in different output data. The first PSE in this paper consists of a classical benchmark problem [2] that involves studying a plane strain plate with a central circular hole. The dimensions of the plate were 18×10 m, with $R = 5$ m. The geometry of the plate, the spatial discretization scheme, and boundary conditions used in the numerical simulations are shown in Figure 1. The 3D finite element mesh employed had 1,152 elements. To generate the PSE tasks, a material parameter, viscosity η, was selected as the variation parameter. Then, 25 different viscosity values for the η parameter were considered, namely, $x10^y$ Mpa, with $x = 1, 2, 3, 4, 5$, and 7 and $y = 4$, 5, 6, and 7, plus 1.10^8 Mpa. In particular, the variation of a

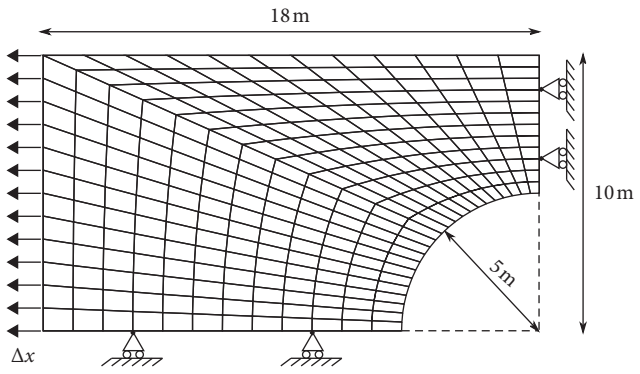

Figure 1: Geometry of the plane strain plate.

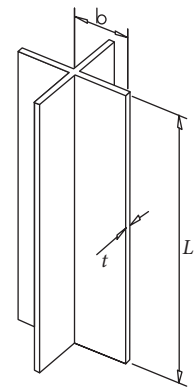

Figure 2: Geometry of the cruciform column.

material parameter in a parametric study may be useful to other disciplines such as design, where it is important that the specialists to know the strength, flexibility, and other characteristics of materials used in the design of certain components.

The second PSE is the study of the elastoplastic buckling behavior of cruciform columns [20], often used in seismic protection of structures as part of energy dissipators. This problem has been taken as a case study to compare the total deformation and incremental theories of plasticity [20, 21]. The geometry of the column used in the numerical simulations is shown in Figure 2. To generate de PSE tasks, 30 different angle values for the α parameter were considered, namely, $\alpha_n = \alpha_{n-1} + 0.25$, with $\alpha_0 = 0.5$ and $n = 1, 2, \ldots, 30$. The variation of an angle parameter in a parametric study may be useful to other disciplines such as seismic protection, where it is important to know the sensitivity to the size of the imperfection.

The third PSE is derived from the two above described PSE applications. We called this application as *ensemble* of PSEs and consists of sending to execute a set of tasks belonging to both PSEs at the same time, which is a realistic Cloud usage scenario.

For evaluating the performance of the autoscaler based on each studied algorithm, we have defined three PSE applications named Plate3D, Cruciform, and Ensemble, each one with three different sizes *small*, *medium*, and *large* according to the number of tasks to execute. For Plate3D and Cruciform, the sizes are 30, 100, and 300 tasks, and for the Ensemble PSE, the number of tasks is 60, 200, and 600. The more the tasks, the deeper the parameter exploration performed by the different PSEs.

5.2. On-Demand and Spot Instances. Characteristics of each type of on-demand instance considered during the experimentation are described in Table 1 and correspond to actual characteristics of Amazon Elastic Compute Cloud (EC2) on-demand instances. Second column shows the number of (virtual) CPUs available for such instance type (vCPU). Then, ECUtot (acronym for EC2 Compute Units) and ECU are the relative computing power of the instances considering all the virtual CPUs and the relative performance of one of the CPUs, respectively. Last column denotes the price

Table 1: Characteristics of the on-demand instances considered. The data correspond to instances belonging to the US-west region.

VM type	νCPU	ECUtot	ECU	Price (USD)
t2.micro	1	1	1	0.013
m3.medium	1	3	2	0.07
c3.2xlarge	8	28	3.5	0.42
r3.xlarge	4	13	3.25	0.35
m3.2xlarge	8	26	3.25	0.56

in US dollars (USD) of an hour of computation. These instance types were selected to provide a diverse spectrum of performance and price configurations.

Spot instances used were set using actual data of Amazon EC2 spot prices corresponding to the US-west region (Oregon) in a tree-month period (between March 7th and June 7th of 2016.). To compute the probabilities of OOB errors, we used the first two months of data by counting the number of times a sliding time window of 1 h (minimum purchase unit in EC2) presented spot prices over the set of possible bid values in x^b. Then, the data corresponding to the last month of the history were kept for the experiments presented in Section 6. This allows us to evaluate the bidding methods and the performance of the autoscaler completely ignoring future evolutions of spot prices, which is the real scenario for any EC2 user.

6. Computational Experiments

We have developed computational experiments to evaluate the autoscaler based on each of the studied algorithms, i.e, NSGA-II and NSGA-III. To perform the experiments, we run the autoscaler endowed with both optimization algorithms. In this section, we describe the procedure used to identify the best parameter setting for NSGA-III. Then, we present the parameterization for NSGA-II. Then, we describe the experimental settings utilized to evaluate the performance of the autoscaler based on each of the studied algorithms. Finally, we present and then analyze the experimental results obtained.

6.1. Parameter Settings for NSGA-III. We developed a sensitivity analysis to identify the best parameter setting for

NSGA-III, regarding each studied application and size. In this analysis, we considered 303 different parameter settings. These settings were generated by the Saltelli sampling method [22], considering the sampling ranges defined for the parameters of NSGA-III, which are presented in Table 2. Then, the NSGA-III was run 30 times for each parameter setting. After each run, the algorithm provided the Pareto set of the last generation. Then, the average value of the hypervolume (HV) measure was calculated on the Pareto sets obtained for each parameter setting. Finally, the parameter setting which maximized the average value of the HV measure was selected as the best setting for NSGA-III. We considered the HV measure since it simultaneously accounts for the closeness of the obtained Pareto sets to the optimal Pareto set and the distribution of the solutions in the obtained Pareto sets.

Table 3 presents the parameter setting selected for NSGA-III regarding each studied application and size, and the corresponding average value of HV. These parameter settings are considered in Section 6.3.

6.2. Parameter Settings for NSGA-II. In relation to the best parameter setting for NSGA-II, regarding each studied application and size, we considered the settings recommended in [11], which were obtained by a sensitivity analysis similar to that developed for NSGA-III. Nevertheless, we defined a new value (i.e., a higher value) for the parameter *evaluations* of NSGA-II in relation to each studied application and size. In this respect, we set the parameter *evaluations* of NSGA-II with the value used for the parameter *evaluations* of the algorithm NSGA-III, for each studied application and size. Thus, the algorithms consider the same termination condition (i.e., the same number of evaluations) in relation to each studied application and size. This was made to guarantee a fair performance comparison between the autoscaler based on NSGA-II and the autoscaler based on NSGA-III.

Table 4 presents the parameter setting for NSGA-II regarding each studied application and size, and the corresponding average value of HV. These parameter settings are considered in Section 6.3.

6.3. Experimental Settings. The autoscaler based on NSGA-II and also the autoscaler based on NSGA-III were evaluated on all the studied applications and sizes. To do that, we used the CloudSim [16], which is heavily utilized by the research community to carry out Cloud experiments.

As NSGA-II and NSGA-III are nondeterministic, each of the mentioned autoscalers was run 30 times on each studied application and size, with the aim of guaranteeing the statistical significance of the results. For each run, the results obtained regarding different metrics including makespan in seconds, cost in USD, and number of task failures were recorded.

In order to develop the runs of the autoscaler based on NSGA-III, we used the parameter settings described in Table 3 for NSGA-III. To develop the runs of the autoscaler based on NSGA-II, we used the parameter settings described in Table 4 for NSGA-II.

6.4. Experimental Results. Table 5 presents the results obtained from the computational experiments. The rows present the results per autoscaler regarding all the studied applications and sizes. For simplicity, in this table and also in this section, the autoscaler based on NSGA-III is called NSGA-III, and the autoscaler based on NSGA-II is called NSGA-II. Columns 4–6 present the average values for the makespan in seconds, cost in USD, and number of task failures, respectively. Column 7 presents the average value for the L_2-norm which considers the makespan, cost, and number of task failures resulting from the experiments. This L_2-norm jointly analyzes the metrics which are interesting for the multiobjective autoscaling problem addressed in this paper.

From Table 5, the following can be mentioned. In relation to the makespan, NSGA-III has obtained a better average performance than NSGA-II in seven of the nine studied applications and sizes, reaching good makespan savings (around 20% in most cases). Regarding the cost, NSGA-III has obtained a much better average performance than NSGA-II in all the studied applications and sizes, reaching very good cost savings (around 50% in some cases and 70% in some other cases). With respect to the average number of task failures, NSGA-III has outperformed NSGA-II in two of the nine studied applications and sizes. However, NSGA-III has obtained an average number of task failures higher than that of NSGA-II in seven of the studied applications and sizes. This is mainly because NSGA-III only used spot instances in such applications and sizes.

Finally, in relation to the L_2-norm, NSGA-III has obtained a much better average performance than NSGA-II in the nine study cases. This is mainly because NSGA-III outperformed NSGA-II in the nine studied cases in terms of cost and outperformed NSGA-II in seven of the studied cases in terms of makespan.

To ascertain the significance of the improvements reached by NSGA-III regarding NSGA-II, we applied a statistical significance test on the results obtained from the experiments for the L_2-norm. Given that the L_2-norm jointly analyzes the makespan, cost, and number of task failures resulting from the experiments, we consider that the L_2-norm is appropriate and useful to develop the statistical significance test. In relation to the results obtained from the experiments for the L_2-norm, each of the autoscalers was run 30 times on each studied application and size. Thus, each autoscaler obtained 30 results for the L_2-norm regarding each studied application and size. We applied the normality Shapiro–Wilk test on the results obtained by each autoscaler regarding each studied application and size, to determine if these results follow a normal distribution or not and decide the statistical significance test to be applied. According to the Shapiro–Wilk test, which was applied with a strong confidence level of $\alpha = 0.001$, the results obtained by each autoscaler regarding each studied application and size do not follow a normal distribution. For this reason, a

Table 2: Sampling ranges for the parameters of NSGA-III.

Parameter	Description	Sampling range
Evaluations	Maximal number of solutions to be evaluated	[500, 20000]
Population	Number of solutions of the population	[90, 200]
Reference points	Number of reference points	[90, 200]
Pc	Crossover probability	[0.8, 1]
Dc	Crossover distribution index	[0, 60]
Pm	Mutation probability	[0.01, 0.6]
Dm	Mutation distribution index	[0, 60]

Table 3: Parameter setting selected for NSGA-III regarding each application and size.

Application	Size	*Eval.*	*Pop.*	*Reference points*	*Pc*	*Dc*	*Pm*	*Dm*	HV
Plate3D	30	18400	92	91	0.81	3.8	0.59	14.6	0.95
	100	18400	92	91	0.92	1.9	0.24	3.1	0.94
	300	18400	92	91	0.83	46.9	0.26	1.5	0.94
Cruciform	30	18400	92	91	0.90	20.7	0.45	5.2	0.81
	100	18400	92	91	0.90	56.3	0.39	7.1	0.80
	300	18400	92	91	0.98	13.3	0.24	3.1	0.80
Ensemble	60	18400	92	91	0.92	13.3	0.24	3.1	0.88
	200	18400	92	91	0.87	46.9	0.26	1.5	0.88
	600	18400	92	91	0.87	28.3	0.09	1.5	0.87

Table 4: Parameter setting considered for NSGA-II regarding each application and size.

Application	Size	*Eval.*	*Pop.*	*Pc*	*Dc*	*Pm*	*Dm*	HV
Plate3D	30	18400	116	0.88	18.9	0.55	10.8	0.87
	100	18400	116	0.88	18.9	0.55	10.8	0.90
	300	18400	172	0.89	13.3	0.57	12.7	0.90
Cruciform	30	18400	116	0.88	18.9	0.55	10.8	0.38
	100	18400	136	0.88	18.9	0.55	10.8	0.57
	300	18400	116	0.88	18.9	0.55	10.8	0.68
Ensemble	60	18400	116	0.88	18.9	0.55	10.8	0.72
	200	18400	116	0.88	18.9	0.55	10.8	0.79
	600	18400	116	0.88	18.9	0.55	10.8	0.80

nonparametric statistical significance test is required in this case. In this respect, we applied the Mann–Whitney U test [23] on the results obtained by the two autoscalers regarding each studied application and size. According to the Mann–Whitney U test, which was applied with a strong confidence level of $\alpha = 0.001$, NSGA-III reached significant improvements in terms of the L_2-norm, in all the studied applications and sizes.

In addition to the results presented in Table 5 and previously analyzed, Table 6 presents the computation time (in seconds) required by each one of the autoscalers, regarding each of the studied applications and sizes. In this table, columns 3 and 6 present the average computation time required by the autoscalers NSGA-II and NSGA-III, respectively. Then, columns 4 and 7 detail the maximal computation time obtained by the autoscalers NSGA-II and NSGA-III, respectively. Finally, columns 5 and 8 present the minimal computation time obtained by the autoscalers NSGA-II and NSGA-III, respectively.

As shown in Table 6, the average computation time required by the autoscaler NSGA-III has not exceeded to

Table 5: Results obtained from the computational experiments. The rows present the results per autoscaler regarding all the studied applications and sizes. For the average makespan, cost, number of task failures, and L_2-norm, lower values represent better results.

Application	Size	Autoscaler	Makespan	Cost	Task failures	L_2-norm
Plate3D	30	NSGA-III	**1142.11**	**0.06**	5.80	**0.26**
		NSGA-II	1381.02	0.23	**4.53**	0.44
	100	NSGA-III	1910.03	**0.12**	**12.05**	**0.17**
		NSGA-II	**1836.50**	0.36	25.77	0.28
	300	NSGA-III	2894.53	**0.39**	**62.32**	**0.23**
		NSGA-II	**2720.12**	0.98	82.17	0.37
Cruciform	30	NSGA-III	**3011.33**	**0.26**	44.75	**0.30**
		NSGA-II	3599.30	0.47	**15.80**	0.42
	100	NSGA-III	**2925.29**	**0.77**	146.33	**0.29**
		NSGA-II	3602.92	0.91	**59.92**	0.46
	300	NSGA-III	**4223.02**	**2.22**	309.80	**0.18**
		NSGA-II	5118.21	4.74	**219.83**	0.30
Ensemble	60	NSGA-III	**5095.71**	**0.26**	35.32	**0.30**
		NSGA-II	5504.60	0.69	**23.33**	0.46
	200	NSGA-III	**5868.27**	**0.91**	147.00	**0.34**
		NSGA-II	6912.26	1.78	**118.07**	0.54
	600	NSGA-III	**5840.39**	**2.80**	372.86	**0.31**
		NSGA-II	7401.15	4.59	**281.16**	0.40

that of the autoscaler NSGA-II, in most applications and sizes studied. It is worth mentioning that the average computation time required by the first step of the autoscalers (i.e., computation time required by the algorithms NSGA-III and NSGA-II) is a small percentage (i.e., no more than 8%) of the computation time corresponding to each autoscaling stage. As was mentioned in Section 2, each autoscaling stage requires 1 hour. It is necessary to mention that all computational experiments were developed on a computer AMD Ryzen 5 2600X Six-Core Processor, clock speed of 2022 Mhz

Table 6: Computation time (in seconds) required by each autoscaler regarding all the studied applications and sizes.

Application	Size	NSGA-II			NSGA-III		
		Average	Max	Min	Average	Max	Min
Plate3D	30	6.50	9	4	**5.13**	8	4
	100	21.13	25	14	**19.79**	28	15
	300	92.63	115	60	**86.06**	119	62
Cruciform	30	6.14	10	5	**5.60**	7	5
	100	24.00	31	17	24.00	32	16
	300	128.14	154	82	**101.38**	121	75
Ensemble	60	13.00	17	9	**11.00**	16	8
	200	**52.40**	65	39	64.30	75	45
	600	**203.86**	235	187	209.56	282	182

per physical core, 16 GB of RAM memory, Solid State Disk 120 GB, and operative system Manjaro (kernel 4.19.6). Besides, the autoscalers were implemented in Java programming language.

From the analysis of the results in Tables 5 and 6, NSGA-III reached significant improvements regarding NSGA-II in all the studied applications and sizes, without exceeding the computation time of NSGA-II in most applications and sizes studied.

6.4.1. Pareto Sets. In this section, we analyze the quality of the Pareto sets obtained by the autoscalers for each of the studied applications and sizes. We focus the attention on the Pareto sets obtained by the autoscalers during the first autoscaling stage (the first hour). This is because the following reason. For each of the studied applications and sizes, during the first autoscaling stage, the autoscalers address exactly the same multiobjective problem and generate Pareto sets for such problem. Thus, it is appropriate and valuable to compare these Pareto sets. In each of the next autoscaling stages (after the first hour), the autoscalers usually address different multiobjective problems. This is mainly because the problems inherent to each of these stages vary according to the state of the PSE' tasks execution and the state of the virtual infrastructure. Therefore, it is not appropriate to compare the Pareto sets obtained for such problems.

To analyze the quality of the Pareto sets, we use two well-known measures which are usually utilized to evaluate Pareto sets obtained by evolutionary algorithms. One of these measures is the hypervolume [15], which calculates the percentage of the objective space volume that is dominated or covered by a given Pareto set and simultaneously accounts for the proximity to the optimal Pareto set and the distribution of the solutions in the Pareto set on the objective space. The other measure is the coverage [15], which calculates the percentage of solutions in a given Pareto set that are dominated by one or more solutions in other given competing Pareto sets.

Table 7 presents the hypervolume (average hypervolume value) and the coverage (average coverage value) of the Pareto sets obtained by NSGA-III and NSGA-II for each one of the studied applications and sizes. Note that the coverage value corresponding to the Pareto sets obtained by NSGA-III refers to the percentage of solutions in these sets that are dominated by solutions in the Pareto sets obtained by NSGA-II. Unlike this, the coverage value corresponding to the Pareto sets obtained by NSGA-II refers to the percentage of solutions in these sets that are dominated by solutions in the Pareto sets obtained by NSGA-III. For the coverage measure, lower values represent better results.

As shown in Table 7, NSGA-III has obtained an average hypervolume value higher than that obtained by NSGA-II, for each of the studied applications and sizes. This means that the objective space volume dominated by the Pareto sets obtained by NSGA-III is larger than that of the Pareto sets obtained by NSGA-II. In this case, the objective space volume is bounded by a reference point which is composed of the maximum value (i.e., the worst value) of each one of the three objectives considered (i.e., makespan, cost, and task failures). Figures 3–5 show the evolution of the average hypervolume value of the Pareto sets obtained by NSGA-III and NSGA-II over the generations, for each application and size. These figures indicate that the Pareto sets obtained by NSGA-III reached better hypervolume values in a less number of generations (i.e, less computation time), for all the studied applications and sizes.

Table 7 also shows that NSGA-III has obtained an average coverage value significantly lower than that obtained by NSGA-II, for each of the studied applications and sizes. In this respect, most solutions in the Pareto sets obtained by NSGA-III (more than 75% of the solutions) are not dominated by solutions in the Pareto sets obtained by NSGA-II, for all applications and sizes. Unlike this, a good number of solutions in the Pareto sets obtained by NSGA-II (more than 50% of the solutions) are dominated by solutions in the Pareto sets obtained by NSGA-III, for all applications and sizes. Figures 6 and 7 show examples of Pareto sets obtained by NSGA-III and NSGA-II for Plate3D with 30 tasks and Ensemble with 60 tasks, respectively. In both cases, as indicated by the average coverage values obtained, most solutions in the Pareto set obtained by NSGA-III (more than 80% of the solutions) are not dominated by solutions in the Pareto set obtained by NSGA-II, and a significant number of solutions in the Pareto set obtained by NSGA-II are dominated by solutions in the Pareto set obtained by NSGA-III. Thus, the solutions in the Pareto set obtained by NSGA-III are distributed closer to the optimal Pareto set than those of the Pareto set obtained by NSGA-II.

The results detailed in Table 7 and previously analyzed indicate that the Pareto sets obtained by NSGA-III have outperformed to those obtained by NSGA-II, in terms of both closeness to the optimal Pareto set and distribution of the solutions on the objective space, for all studied applications and sizes.

7. Related Work

Cloud autoscaling involves solving two different complex optimization problems known as scaling and scheduling. The efficient management of scientific applications on the Cloud via autoscaling techniques is an important problem [21], and hence, many approaches have tackled the tasks

Table 7: Hypervolume (average hypervolume value) and coverage (average coverage value) of the Pareto sets obtained by each autoscaler during the first autoscaling stage (first hour), for each studied application and size. For hypervolume measure, higher values represent better results. For coverage measure, lower values represent better results.

Application	Size	Hypervolume (%)		Coverage (%)	
		NSGA-III	NSGA-II	NSGA-III	NSGA-II
Plate3D	30	**88**	81	**19.10**	53.51
	100	**87**	84	**21.35**	55.26
	300	**87**	85	**24.72**	57.89
Cruciform	30	**22**	16	**5.56**	53.57
	100	**21**	17	**8.33**	55.95
	300	**21**	17	**12.50**	59.52
Ensemble	60	**39**	34	**17.98**	52.63
	200	**41**	37	**20.22**	54.39
	600	**41**	38	**23.60**	57.02

scheduling part of the problem via single-objective [24–26] and multiobjective scheduling approaches [27, 28]. However, there are few efforts that deal with the autoscaling problem as a whole, scaling and scheduling, by using strategies based on a multiobjective metaheuristic which consider the use both of on-demand and spot instances while minimizing the makespan and the monetary cost of PSE applications. Only in a previous work of some of the authors of this work [11], an autoscaler called MOEA which considers spot instances was proposed. Although one of the objectives in [11] was to minimize the failures produced for the use of spot instances, they were not completely reduced. The unexpected completion of certain instances impacts their associated task finish time since these tasks must be scheduled in other instances. A major distinction of this paper with respect to [11] is that MOEA implements the NSGA-II algorithm, while in this work we perform the scaling and scheduling stages using NSGA-III, obtaining as reported earlier noticeable performance gains.

There are other works where spot instances were also considered. Among them, we can mention the work in [5] where the authors proposed an autoscaler called SIAA for workflow applications with the aim of minimizing the makespan subject to budget constraints. The main difference concerning this work is that in [5], the monetary cost was not considered. Then, in [9], a cost-efficient based scheduling algorithm that allows leasing instances from Clouds for executing scientific workflows while meeting the required deadlines of tasks was proposed. The tasks are scheduled according to the spot instance pricing. On the other hand, the work in [29] is focused on running large-scale computational applications on Clouds, especially for on-demand and spot instances offered by Amazon EC2. In [29], after analyzing the characteristic of the spot price and the effect of spot instances disturbance, the authors proposed a dynamic approach to reduce cost, increasing the reliability and reducing the complexity of fault tolerance without affecting the overall performance and scalability. The main difference between the works [9, 29] and ours is that we focus on a budget-constrained

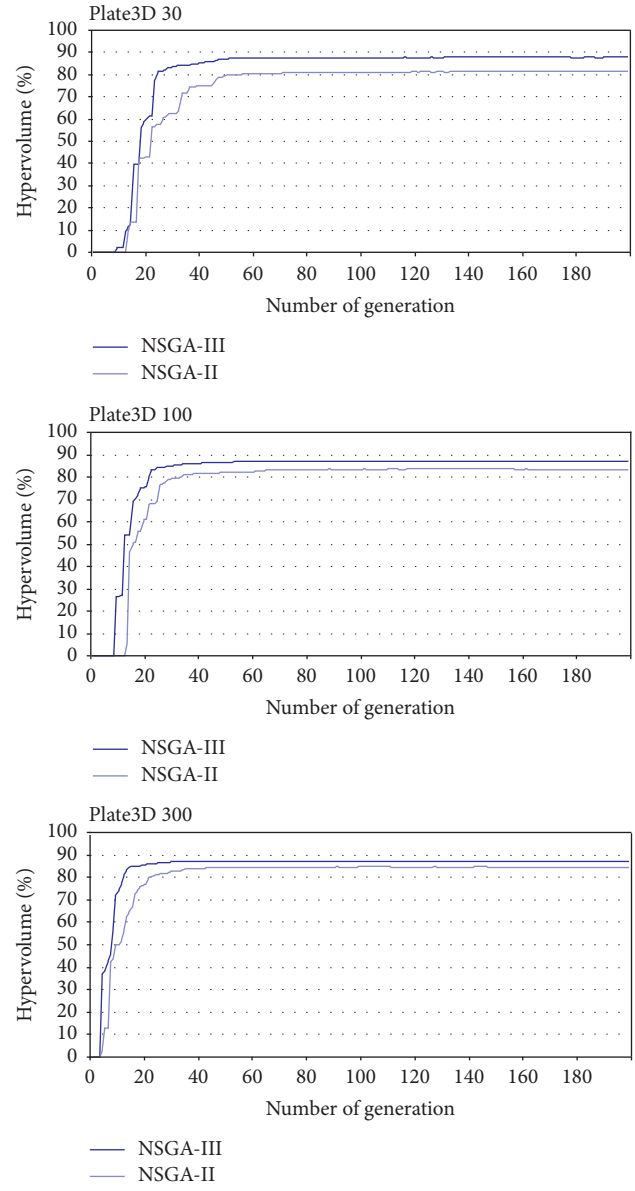

Figure 3: Evolution of the hypervolume value of the Pareto sets obtained by NSGA-III and NSGA-II over the generations, for Plate3D with 30, 100, and 300 tasks.

autoscaling problem while the mentioned works focus on solving scheduling problems subject to task deadline constraints; thus, they are useful in different scenarios. Another important distinction is that we are focused on scientific PSEs and not workflow applications.

There are other works that differ from ours because they were proposed for addressing workflow autoscaling [6, 7, 10] with deadline or budget constraints. First, in [6], the authors proposed an autoscaling strategy for the efficient execution of multiple workflow applications subject to deadline constraints. The goal was to ensure that all tasks finish before their respective deadlines by using the cheapest resources whenever possible. Later, the same authors moved to the problem of workflow autoscaling but considering budget constraints [7]. Then, is the work presented in [10], where

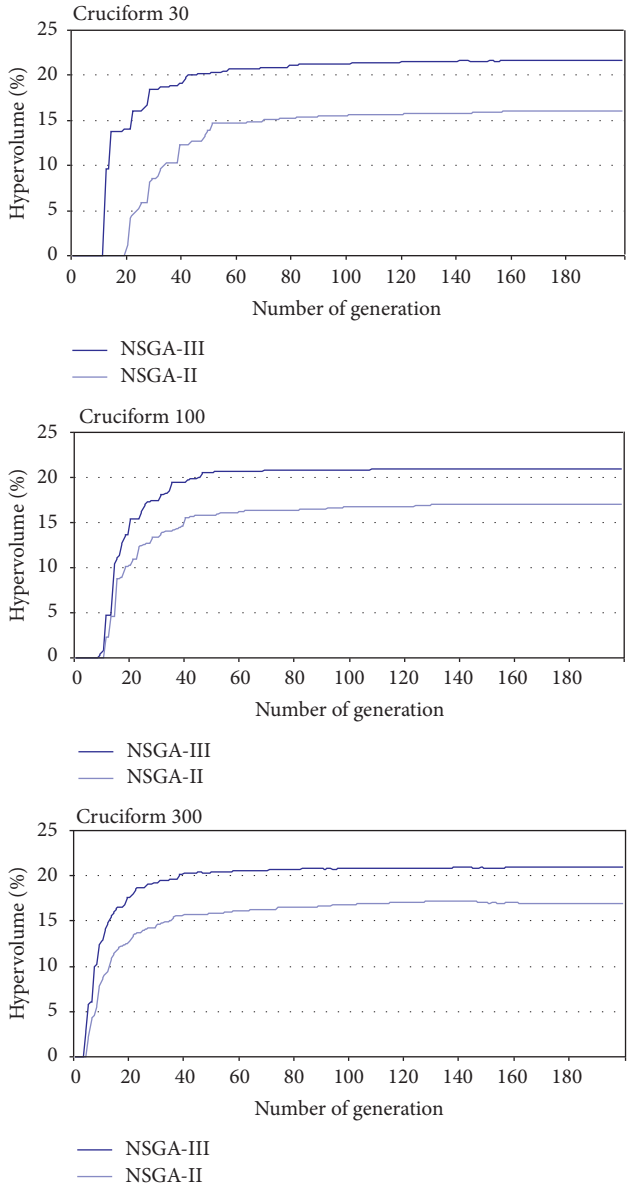

Figure 4: Evolution of the hypervolume value of the Pareto sets obtained by NSGA-III and NSGA-II over the generations, for Cruciform with 30, 100, and 300 tasks.

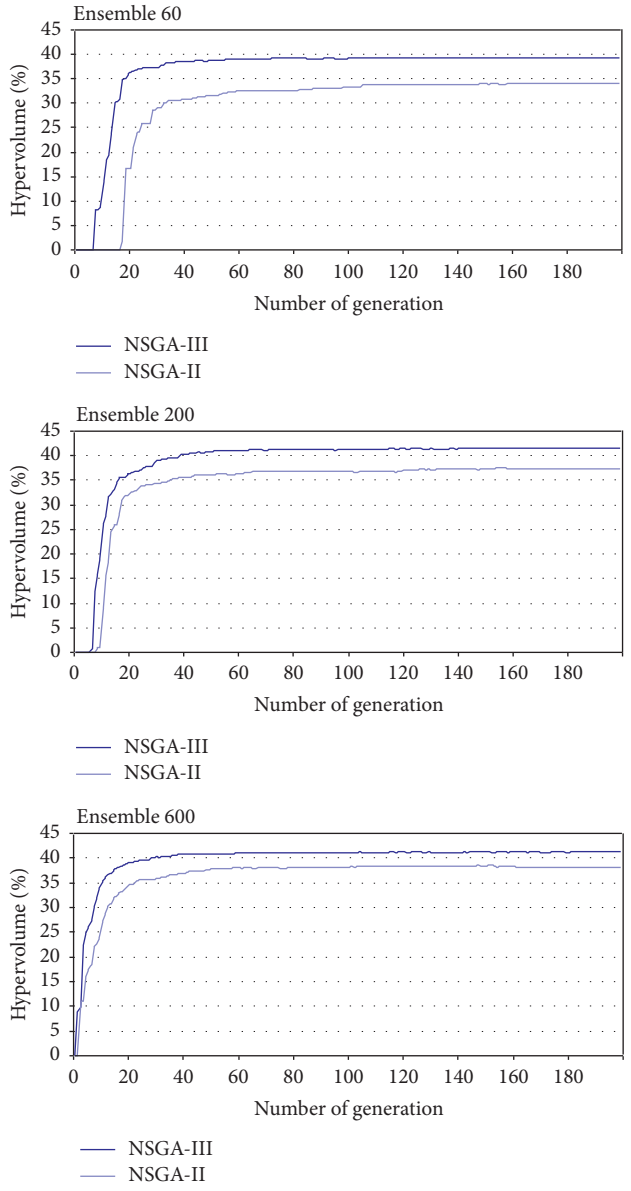

Figure 5: . Evolution of the hypervolume value of the Pareto sets obtained by NSGA-III and NSGA-II over the generations, for Ensemble with 60, 200, and 600 tasks.

the main characteristics of the tasks in the workflow structure are learned over time, i.e., the autoscaler dynamically adapts the number of allocated resources in order to meet the deadlines of all tasks without knowing the workflow structure itself and without any information of the execution time. The goal of this work was to minimize the makespan and cost.

Other works are proposed in [30, 31]. In [30], a reliable autoscaling algorithm for web applications using heterogeneous spot instances along with on-demand instances was proposed. The idea of this work was to take advantage of different prices among various types of spot instances to reach both high availability, monetary cost saving, and low response time, even when some types of spot instances are terminated unexpectedly by the Cloud provider. For this, the authors have implemented a fault-tolerant mechanism to further overprovision the same amount of capacity using another type of spot instance. In this way, the application can tolerate the termination of any involving type of VMs and remain fully provisioned. Then, in [31], the authors proposed RLPAS, a reinforcement learning approach to automatically scale virtualized resources in the Cloud. The purpose is to dynamically scale the resources to minimize response time while maximizing resource utilization and throughput. The RLPAS was proposed for learning the environment in parallel where the workloads are heterogeneous and fluctuating and the Cloud instances are on-demand. As can be seen in [30, 31], the algorithms were implemented for web applications where task requirements are much lighter compared to the applications, i.e., PSEs, which we consider in this work.

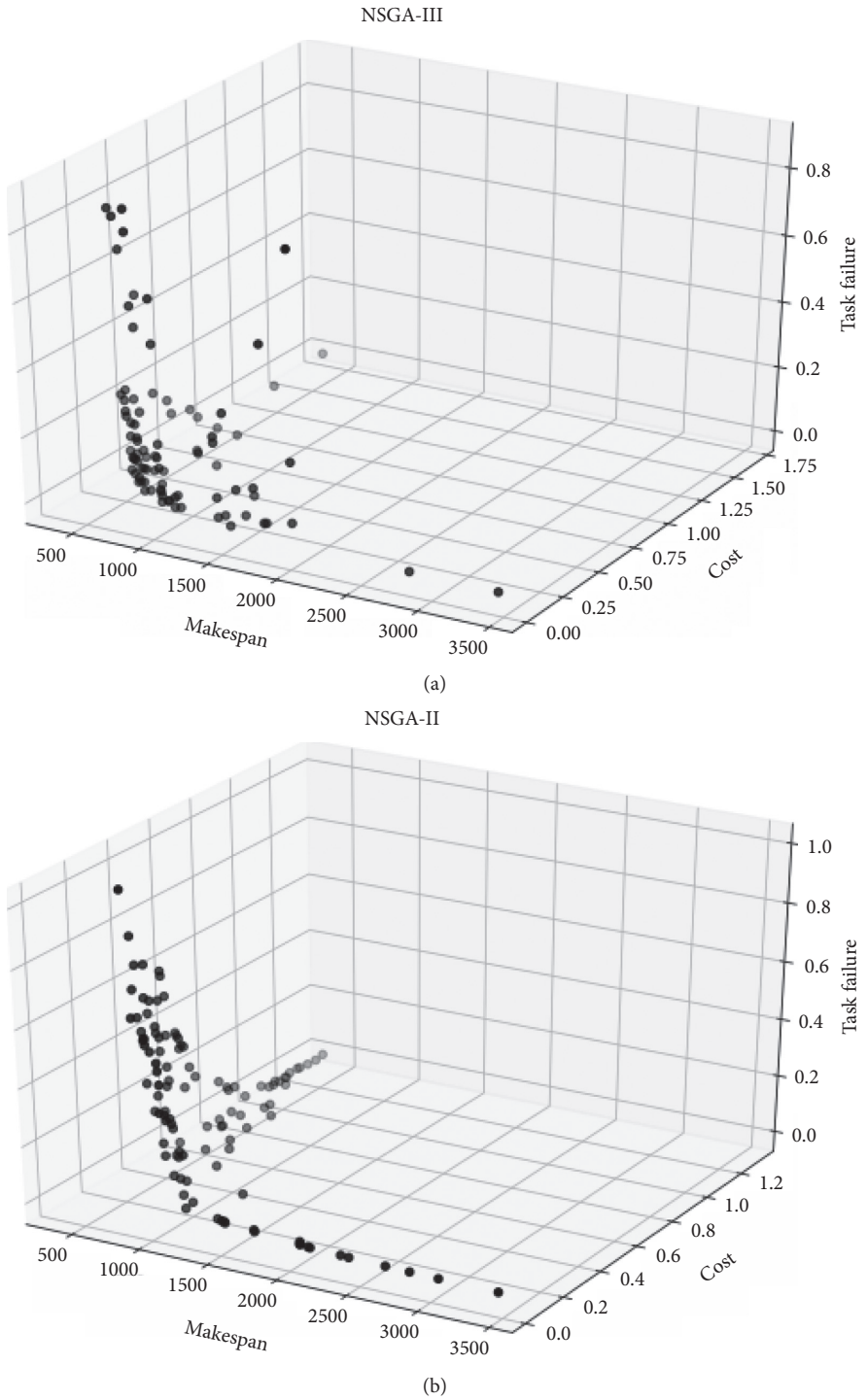

Figure 6: Pareto sets obtained by NSGA-III and NSGA-II for Plate3D with 30 tasks.

On the other hand, in [32], a strategy that produces elastic clusters from the computational resources provided by multiple Clouds was presented. Particularly, the work is focused on hybrid clusters across on-premises and public Clouds and the use of Amazon spot instances to achieve reliable low cost. In order to achieve these goals, the authors implemented a check-pointing algorithm which allows tasks to periodically save the progress made before the spot instance is finished by the provider, thereby facilitating task resumption from the last checkpoint. The authors have performed a case study based on a scientific application implemented in MPI for the nonlinear dynamic analysis of buildings. Then, in [8], the authors have proposed a dynamic Cloud resource provisioning called delay-based dynamic scheduling (DDS) to minimize the monetary cost while meeting Bag-of-Tasks (BoT) workflow deadlines, i.e., new

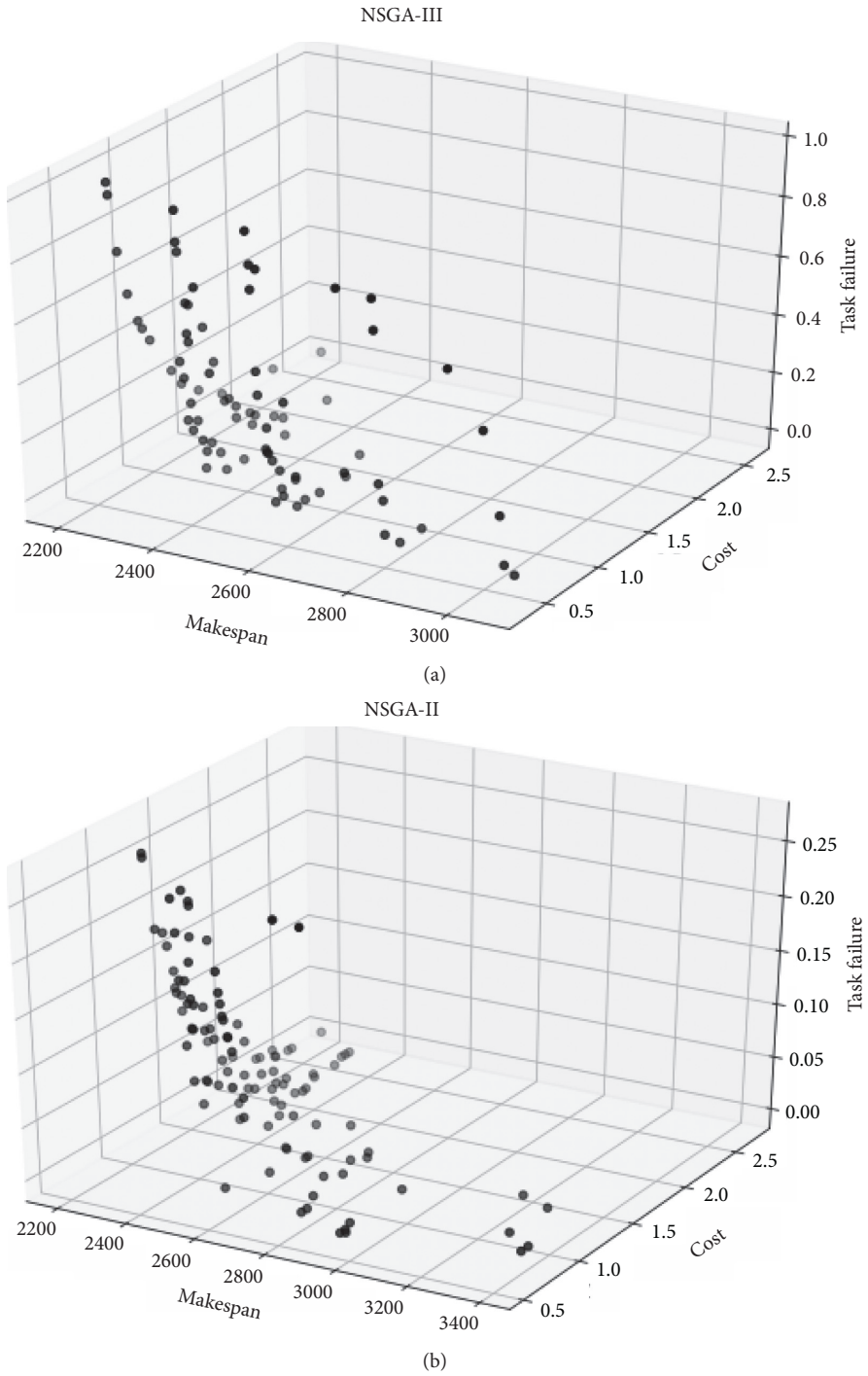

Figure 7: Pareto sets obtained by NSGA-III and NSGA-II for Ensemble with 60 tasks.

VMs are dynamically rented by the DDS according to the execution state and the estimated task execution times to fulfill the application deadline. It is important to mention that although the approaches consider applications with independent tasks and the monetary costs, the algorithms in [8, 32] are not based on metaheuristics as we propose in this work.

There are also two works that propose the use of the NSGA-III algorithm [33, 34]. In [34], the authors have proposed a multiobjective optimization algorithm based on NSGA-III for the execution of workflow applications in Cloud. The goal of this work is to minimize the cost and makespan; at the same time, the VM utilization is maximized. Then, in [33], a multiobjective scheduling algorithm based on E-NSGA-III for workflow applications has been proposed. In this approach, E-NSGA-III utilizes extreme solutions in the population generation module in order to improve quality in terms of makespan and cost. Although both works propose

the use of algorithms based on NSGA-III, both approaches focus only on task scheduling and do not take into account the variations on the virtual infrastructure availability to rescheduling the execution of tasks every certain periods of time. In addition, both works focus on the execution of workflow applications and not PSEs. Besides, in none of the works [33, 34], the authors considered the use of spot instances such we propose in this work.

It is worth noting that, from the related works found, most of them have been proposed for workflows considering task deadlines or budget constraints, and only in two works [10, 30], in addition to our previous work [11], the authors have proposed to minimize the makespan, rendering difficulty of their applicability to execute scientific applications, such as PSEs, in Clouds infrastructures. In this context, in [35], the approach minimizes cost in Amazon infrastructure, regardless of spots. In contrast to our approach, the authors consider task deadlines instead of budget constraints. Moreover, the nonautoscaled approach adopted optimizes resource usage by utilizing mixed integer programming, instead of evolutionary algorithms. In line with our approach, the authors in [36] minimize time and cost, but they do not consider spot instances. Moreover, their approach fails to be a pure autoscaler. Prepared to be useful for workflows, the approach explores NSGA-II instead of NSGA-III.

Besides, another distinction is that only in two of the surveyed works [33, 34], the authors have considered the use of the NSGA-III metaheuristic. However, both approaches have been proposed for task scheduling without considering the automatic, dynamic scaling of the infrastructure. Concretely, in this work, the objectives are to minimize the failure probability as well as the makespan and the monetary cost when different types of on-demand and spot instances are considered.

8. Conclusions

We addressed the problem of autoscaling PSEs in public Clouds (e.g., Amazon EC2 and Google Cloud), considering that the instances of VMs can be acquired under two possible pricing models: the on-demand model and the spots model. This problem implies determining the number and type of on-demand and spot instances to be acquired for executing the tasks in a PSE, and the bid prices corresponding to the spot instances, so that the predetermined optimization objectives are reached. In this respect, three relevant optimization objectives were considered: the minimization of the makespan, the monetary cost, and the impact of OOB errors which are inherent to spot instances. To solve the resulting optimization problem, the well-known multiobjective genetic algorithms NSGA-II and NSGA-III were exploited as part of the autoscaler MOEA, which is the focus of this paper.

The autoscaler endowed with both algorithms was evaluated on three different real-world PSEs, considering three different sizes (i.e., numbers of tasks) for each PSE. Moreover, different types of on-demand and spot instances available in Amazon EC2 were considered. These different instance types have different characteristics in relation to the processing capacity and also differ regarding the monetary cost. These PSE applications and instance types were considered in order to provide diverse realistic experimental settings. Then, the performance of the autoscaler based on NSGA-III was compared in detail with that of the autoscaler based on NSGA-II.

According to the performance comparison carried out, the autoscaler based on NSGA-III has significantly outperformed the autoscaler based on NSGA-II in terms of the L_2-norm which jointly assesses makespan, cost, and number of task failures caused by OOB errors, in all the PSE applications and sizes considered. Thus, we can mention that NSGA-III may be considered as a better alternative than NSGA-II for solving different instances of the multiobjective autoscaling problem addressed in this paper. Future work will explore other realistic scenarios, including simultaneous optimized autoscaling of PSEs belonging to different users, as well as federated Clouds in addition to single Clouds.

Data Availability

The spot prices data used to support the findings of this study have been deposited in the Amazon Web Services (AWS) Spot Prices Data 2016, Mendeley Data, v1 (https://doi.org/10.17632/zcnp5xwvz6.1).

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

The authors acknowledge the financial support by CONICET, grant no. 11220170100490CO, and the SIIP-UNCuyo, project no. B082.

References

[1] V. Mauch, M. Kunze, and M. Hillenbrand, "High performance cloud computing," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1408–1416, 2013.

[2] C. García Garino, M. S. Ribero Vairo, S. Andía Fagés, A. E. Mirasso, and J.-P. Ponthot, "Numerical simulation of finite strain viscoplastic problems," *Journal of Computational and Applied Mathematics*, vol. 246, pp. 174–184, 2013.

[3] E. Pacini, C. Mateos, C. G. Garino, C. Careglio, and A. Mirasso, "A bio-inspired scheduler for minimizing makespan and flowtime of computational mechanics applications on federated clouds," *Journal of Intelligent & Fuzzy Systems*, vol. 31, no. 3, pp. 1731–1743, 2016.

[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[5] D. A. Monge, Y. Garí, C. Mateos, and C. García Garino, "Autoscaling scientific workflows on the cloud by combining on-demand and spot instances," *Computer Systems Science and Engineering*, vol. 32, no. 4, pp. 291–306, 2017.

[6] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proceedings of the International Conference for High*

Performance Computing, Networking, Storage and Analysis (SC), pp. 1–12, Seattle, WA, USA, November 2011.

[7] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in *Proceedings of the 27th International Symposium on Parallel & Distributed Processing*, pp. 67–78, IEEE Computer Socienty, Boston, MA, USA, May 2013.

[8] Z. Cai, X. Li, R. Ruiz, and Q. Li, "A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds," *Future Generation Computer Systems*, vol. 71, pp. 57–72, 2017.

[9] J. Li, S. Su, X. Cheng, M. Song, L. Ma, and J. Wang, "Cost-efficient coordinated scheduling for leasing cloud resources on hybrid workloads," *Parallel Computing*, vol. 44, pp. 1–17, 2015.

[10] E. De Coninck, T. Verbelen, B. Vankeirsbilck, S. Bohez, P. Simoens, and B. Dhoedt, "Dynamic auto-scaling and scheduling of deadline constrained service workloads on iaas clouds," *Journal of Systems and Software*, vol. 118, pp. 101–114, 2016.

[11] D. A. Monge, E. Pacini, C. Mateos, and C. García Garino, "Meta-heuristic based autoscaling of cloud-based parameter sweep experiments with unreliable virtual machines instances," *Computers & Electrical Engineering*, vol. 69, pp. 364–377, 2018.

[12] G. Campos Ciro, F. Dugardin, F. Yalaoui, and R. Kelly, "A nsga-II and nsga-III comparison for solving an open shop scheduling problem with resource constraints," *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 1272–1277, 2016.

[13] J.-H. Yi, S. Deb, J. Dong, A. H. Alavi, and G.-G. Wang, "An improved nsga-III algorithm with adaptive mutation operator for big data optimization problems," *Future Generation Computer Systems*, vol. 88, pp. 571–585, 2018.

[14] P. Wangsom, K. Lavangnananda, and P. Bouvry, "Multi-objective scheduling for scientific workflows on cloud with peer-to-peer clustering," in *Proceedings of the 11th International Conference on Knowledge and Smart Technology (KST)*, pp. 175–180, Phuket, Thailand, April 2019.

[15] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, Springer, Berlin, Germany, 2nd edition, 2015.

[16] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: nsga-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[18] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based non-dominated sorting approach, part I: solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2014.

[19] I. Das and J. E. Dennis, "Normal-boundary intersection: a new method for generating the pareto surface in nonlinear multicriteria optimization problems," *SIAM Journal on Optimization*, vol. 8, no. 3, pp. 631–657, 1998.

[20] N. Makris, "Plastic torsional buckling of cruciform compression members," *Journal of Engineering Mechanics*, vol. 129, no. 6, pp. 689–696, 2003.

[21] M. A. Netto, C. Cardonha, R. L. Cunha, and M. D. Assuncao, "Evaluating auto-scaling strategies for cloud computing environments," in *Proceedings of the 22nd International Symposium on Modelling, Analysis Simulation of Computer and Telecommunication Systems*, pp. 187–196, IEEE Computer Socienty, Paris, France, September 2014.

[22] A. Saltelli, M. Ratto, T. Andres et al., *Global Sensitivity Analysis: The Primer*, John Wiley & Sons, Hoboken, NJ, USA, 2008.

[23] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.

[24] M. Adhikari, S. Nandy, and T. Amgoth, "Meta heuristic-based task deployment mechanism for load balancing in iaas cloud," *Journal of Network and Computer Applications*, vol. 128, pp. 64–77, 2019.

[25] M. Kalra and S. Singh, "A review of metaheuristic scheduling techniques in cloud computing," *Egyptian Informatics Journal*, vol. 16, no. 3, pp. 275–295, 2015.

[26] I. Casas, J. Taheri, R. Ranjan, L. Wang, and A. Y. Zomaya, "GA-ETI: an enhanced genetic algorithm for the scheduling of scientific workflows in cloud environments," *Journal of Computational Science*, vol. 26, pp. 318–331, 2018.

[27] H. Hu, Z. Li, H. Hu et al., "Multi-objective scheduling for scientific workflow in multicloud environment," *Journal of Network and Computer Applications*, vol. 114, pp. 108–122, 2018.

[28] S. Srichandan, T. Ashok Kumar, and S. Bibhudatta, "Task scheduling for cloud computing using multi-objective hybrid bacteria foraging algorithm," *Future Computing and Informatics Journal*, vol. 3, no. 2, pp. 210–230, 2018.

[29] S. Lu, X. Li, L. Wang et al., "A dynamic hybrid resource provisioning approach for running large-scale computational applications on cloud spot and on-demand instances," in *Proceedings of the International Conference on Parallel and Distributed Systems*, pp. 657–662, Seoul, South Korea, December 2013.

[30] C. Qu, R. N. Calheiros, and R. Buyya, "A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances," *Journal of Network and Computer Applications*, vol. 65, pp. 167–180, 2016.

[31] J. V. BibalBenifa and D. Dejey, "Rlpas: reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment," *Mobile Networks and Applications*, vol. 24, no. 4, pp. 1–16, 2018.

[32] A. Calatrava, E. Romero, G. Moltó, M. Caballer, and J. M. Alonso, "Self-managed cost-efficient virtual elastic clusters on hybrid cloud infrastructures," *Future Generation Computer Systems*, vol. 61, pp. 13–25, 2016.

[33] K. Lavangnananda, P. Wangsom, and P. Bouvry, "Extreme solutions nsga-III (e-nsga-III) for scientific workflow scheduling on cloud," in *Proceedings of the 17th IEEE International Conference on Machine Learning and Applications (ICMLA2018)*, pp. 17–20, Orlando, FL, USA, December 2018.

[34] P. Wangsom, K. Lavangnananda, and P. Bouvry, "The application of nondominated sorting genetic algorithm (nsga-III) for scientific-workflow scheduling on cloud," in *Proceedings of the 8th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2017)*, pp. 269–287, Kuala Lumpur, Malaysia, December 2017.

[35] M. Malawski, K. Figiela, M. Bubak, E. Deelman, and J. Nabrzyski, "Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization," *Scientific Programming*, vol. 2015, Article ID 680271, 13 pages, 2015.

[36] P. T. Thant, C. Powell, M. Schlueter, and M. Munetomo, "Multiobjective level-wise scientific workflow optimization in iaas public cloud environment," *Scientific Programming*, vol. 2017, 17 pages, 2017.