Hindawi
Scientific Programming
Volume 2023, Article ID 8345646, 26 pages
https://doi.org/10.1155/2023/8345646

Research Article

An In-depth Benchmarking of Evolutionary and Swarm Intelligence Algorithms for Autoscaling Parameter Sweep Applications on Public Clouds

Virginia Yannibelli,[1] **Elina Pacini**,[2,3,4] **David A. Monge**,[2] **Cristian Mateos**,[1] **Guillermo Rodriguez**,[1] **Emmanuel Millán**,[2,4,5] **and Jorge R. Santos**[5]

[1]*ISISTAN (UNICEN-CONICET), Tandil (7000), Buenos Aires, Argentina*
[2]*ITIC, UNCuyo, Mendoza, Argentina*
[3]*Facultad de Ingeniería, UNCuyo, Mendoza, Argentina*
[4]*CONICET, Buenos Aires, Argentina*
[5]*Facultad de Ciencias Exactas y Naturales, UNCuyo, Mendoza, Argentina*

Correspondence should be addressed to Virginia Yannibelli; virginiayannibelli@gmail.com

Received 15 July 2022; Revised 7 November 2022; Accepted 15 January 2023; Published 17 February 2023

Academic Editor: Zhihan Liu

Many important computational applications in science, engineering, industry, and technology are represented by PSE (parameter sweep experiment) applications. These applications involve a large number of resource-intensive and independent computational tasks. Because of this, cloud autoscaling approaches have been proposed to execute PSE applications on public cloud environments that offer instances of different VM (virtual machine) types, under a pay-per-use scheme, to execute diverse applications. One of the most recent approaches is the autoscaler MOEA (multiobjective evolutive algorithm), which is based on the multiobjective evolutionary algorithm NSGA-II (nondominated sorting genetic algorithm II). MOEA considers on-demand and spot VM instances and three optimization objectives relevant for users: minimizing the computing time, monetary cost, and spot instance interruptions of the application's execution. However, MOEA's performance regarding these optimization objectives depends significantly on the optimization algorithm used. It has been shown recently that MOEA's performance improves considerably when NSGA-II is replaced by a more recent algorithm named NSGA-III. In this paper, we analyze the incorporation of other multiobjective optimization algorithms into MOEA to enhance the performance of this autoscaler. First, we consider three multiobjective optimization algorithms named E-NSGA-III (extreme NSGA-III), SMS-EMOA (S-metric selection evolutionary multiobjective optimization algorithm), and SMPSO (speed-constrained multiobjective particle swarm optimization), which have behavioral differences with NSGA-III. Then, we evaluate the performance of MOEA with each of these algorithms, considering the three optimization objectives, on four real-world PSE applications from the meteorology and molecular dynamics areas, considering different application sizes. To do that, we use the well-known CloudSim simulator and consider different VM types available in Amazon EC2. Finally, we analyze the obtained performance results, which show that MOEA with E-NSGA-III arises as the best alternative, reaching better and significant savings in terms of computing time (10%–17%), monetary cost (10%–40%), and spot instance interruptions (33%–100%).

1. Introduction

Many important scientific applications are represented by PSE (parameter sweep experiment) [1]. A PSE application is defined with the aim of exploring the different behaviors of a determined computational model, where such behaviors are obtained by varying the model's parameter settings. For example, the elastoplastic buckling behavior of a cruciform column can be explored by executing the column's computational model many times, each of them with a different

parameter setting for the model, where the model-specific parameters include column length and width, column thickness, and column cross-section angle [2]. This particular PSE is useful in structural engineering to analyze the stability, strength, rigidity, and earthquake susceptibility of cruciform columns designed for different kinds of structures.

In a PSE, each of the parameters can be assigned different feasible values. Next, the PSE execution includes a number of tasks equal to the number of parameter values that have been varied in the model. Each of these tasks consists of running the model with different parameter values and generating a behavior with respect to the parameter value used. The PSE application results are obtained by executing all the PSE-associated tasks.

PSE tasks are characterized by requiring a considerable number of computer resources and computing time to execute. Besides, PSE tasks are characterized by being totally independent, which means they can be executed in a parallel way. Because of this, PSEs are considered suitable for distributed infrastructures, such as those provided by public clouds [3–5]. In this sense, by executing these applications in cloud environments, the speedup that can be achieved is significant.

Public cloud environments, such as Amazon EC2, Google Cloud, Microsoft Azure, and IBM Cloud, offer their users the possibility of acquiring instances of many different VM (virtual machine) types under a pay-per-use scheme to execute diverse kinds of applications. In this regard, different VM types are composed of different hardware and software configurations, including CPU, memory space, storage space, and operating systems. Besides, different VM types have different monetary costs. In addition, the monetary cost of each VM depends on the pricing model utilized by the users to acquire the instances. In the case of Amazon, two major pricing models exist, on-demand and spot. Under the on-demand model, the instances can be required for a predefined amount of computing time at a predefined monetary cost. For the spot pricing model [6], the instance cost is much lower than that of the instances under the on-demand model, but varies heavily over time according to demand. Besides, the instances under the spot model are subject to interruptions by the cloud service provider (CSP). These interruptions negatively impact the application's tasks running on the instances, and therefore the whole application. Thus, on a public cloud, the computing time and execution monetary cost of an application depends on the number and type of the acquired VMs and the pricing model used to acquire the instances.

Considering these trade-offs, for executing applications in a public cloud environment, it is important to decide the type and number of VMs to be acquired from the cloud service provider, and the pricing model to be used for acquiring the instances of each VM type. Moreover, it is necessary to decide the schedule of the application's tasks on the requested instances. These decisions should be made so that the computational time, the cost, and also the spot instance interruptions are minimized. This problem is treated as a multiobjective NP-hard optimization problem [7].

In the literature, diverse cloud *autoscaling* strategies have been proposed with the aim of automatically and dynamically determining the VM instances to be requested from a CSP for executing a given application in a cloud environment [8–11]. These approaches are mainly characterized by scaling up and down the number of acquired instances over time, considering the application's workload, and scheduling such workload on the acquired instances. However, these approaches differ in many aspects, including the pricing model supported (e.g., only on-demand or both on-demand and spot), the optimization objectives considered, the optimization algorithms applied, and the kind of application for which they were proposed.

As far as we know, a recent cloud autoscaling strategy proposed to execute PSE applications is the MOEA autoscaler [12]. MOEA autoscaler is based on the well-known multiobjective evolutionary algorithm NSGA-II [13], uses spot and on-demand VMs to execute the PSE tasks of a given application, and considers optimization objectives that are very relevant for the users to minimize the computational time, the cost, and the spot instance interruptions of the application's execution. Even though this autoscaler has achieved a good performance in relation to the optimization objectives considered, its performance depends significantly on the Pareto set (i.e., set of non-dominated solutions) provided by the optimization algorithm used. In [14], it has been shown that the performance of this autoscaler improves considerably when the algorithm NSGA-II is replaced by a more recent and well-known multiobjective evolutionary algorithm named NSGA-III [15]. However, the algorithm NSGA-III has limitations in terms of the diversity of the resulting Pareto set [16, 17], which can negatively impact the performance of the autoscaler. Thus, the incorporation of other multiobjective optimization algorithms into this autoscaler could benefit its performance, and therefore, the applicability of this autoscaler to complex real-world PSE applications.

Considering the above mentioned, in this article, we explore the incorporation of other multiobjective optimization algorithms into the autoscaler MOEA with the aim of enhancing the performance of this autoscaler in respect of the optimization objectives considered. Therefore, we implement three new algorithms that have relevant differences in their behavior compared to the algorithm NSGA-III used in [14]. The first algorithm is the multiobjective evolutionary algorithm E-NSGA-III[18, 19], which has been recently proposed in the literature in order to address the limitations of NSGA-III and thus improve the quality (i.e., the diversity, distribution, and convergence) of the Pareto sets generated by NSGA-III. E-NSGA-III has been shown to be more effective than NSGA-III in terms of the quality of the Pareto sets generated, on several NP-hard multiobjective optimization problems, including task scheduling problems in cloud environments [18]. Based on the previously mentioned considerations, we consider that E-NSGA-III could be a valuable alternative for the autoscaler MOEA.

The other algorithms are the multiobjective evolutionary algorithm SMS-EMOA [20] and the multiobjective particle swarm optimization algorithm SMPSO [21]. The algorithm SMS-EMOA is characterized by maximizing the hypervolume of the Pareto set (i.e., the convergence, diversity, and distribution of the Pareto set) as part of the optimization process developed by the algorithm. The algorithm SMPSO is characterized by controlling the generation of candidate solutions for the Pareto set over the search space in order to obtain a diverse and well-distributed Pareto set. As described in [20, 21], due to the mentioned characteristics, both algorithms have been shown to be more effective than well-known multiobjective evolutionary algorithms, such as NSGA-II, in terms of the quality of the Pareto sets obtained on several NP-hard multiobjective optimization problems, including scheduling problems [20, 21]. Taking into consideration that these algorithms have characteristics aimed at generating Pareto sets with high quality, it is worth analyzing empirically the incorporation of these algorithms into the autoscaler MOEA in order to determine if they could be a useful alternative or not for the autoscaler.

To empirically and comparatively measure the MOEA autoscaler performance with the three previously mentioned algorithms, regarding the optimization objectives considered, we use four real-world PSE applications from two different areas, namely, meteorology and molecular dynamics, and also real VM instance data from a real-world cloud environment (i.e., Amazon EC2). The experimental evaluations were conducted using the well-known CloudSim simulator [22].

The remainder of the article is structured as shown below. Then, Section 2 describes the multiobjective cloud autoscaling problem dealt with in this work and also presents its mathematical formulation. Section 3 presents the autoscaler MOEA in detail. Then, Section 4 describes the three considered multiobjective optimization algorithms. In Section 5, the computational experiments performed in order to evaluate the MOEA's performance with the other considered algorithms, and a comparative study of the obtained results, are presented in detail. Next, Section 6 discusses related works. Finally, Section 7 concludes this work and delineates future research lines.

2. Multiobjective Cloud Autoscaling Problem

In this paper, we address the multiobjective cloud autoscaling problem for PSEs. A PSE application is integrated with many computational tasks which are resource intensive and also independent. Because of this, these PSEs are considered suited for public cloud environments. These environments give the possibility of acquiring instances of many diverse VM types. Instances of diverse VM types are set up with different software and hardware configurations and also have different costs. Furthermore, each type of instance can be acquired by the users under different pricing models.

Two pricing models, spot and on-demand, are considered here. For the on-demand pricing model, the instances can be acquired by the users for a predefined amount of computing time at a predefined monetary cost. We will refer to instances acquired under this model as on-demand instances. Then, in the spot model, the instances' cost is lower than that of the on-demand instances, but varies throughout time mainly according to demand. Thus, in order to acquire a spot instance, a user can submit the maximum price that he/she is willing to pay for the instance. This maximum price is here referred to as the bid. Then, while the user's bid is higher than or equal to the current instance cost, the instance will be held for the user. On the other hand, if the cost of the instance varies and exceeds the user's bid, an *interruption* takes place, and the instance is terminated. As a result, the tasks running on the instance are terminated, which is known as task failure. Therefore, spot instance interruptions have a negative impact on the whole application's execution because canceled tasks have to be restarted later. We will refer to acquired instances under this model as spot instances.

The multiobjective cloud autoscaling problem tackled in this work involves two connected problems, as shown in Figure 1. The first of these two problems is defining a scaling plan, which determines the type and number of spot and on-demand instances to be requested from the CSP at the current autoscaling stage for executing the PSE application's tasks. This scaling plan also must determine the bids to be made to the CSP in order to acquire the desired spot instances. The second of these two problems is scheduling the PSE application's tasks on the purchased instances. These two problems must be addressed in order to meet optimization objectives. In this sense, three optimization objectives relevant for the users are regarded as follows: to minimize the makespan, cost, and spot instance interruptions.

Besides, the two problems mentioned must be addressed every a predefined period of time throughout the PSE application's execution. Each one of these periods of time throughout the PSE application's execution is referred to as an autoscaling stage.

Thus, at the beginning of each autoscaling stage, the virtual infrastructure composed by the acquired instances is updated (i.e., scaled up/scaled down) conforming to the application's workload (i.e., the number of the application's pending tasks), and such pending tasks are scheduled on the infrastructure, in such way that the optimization objectives are achieved.

2.1. Mathematical Formulation of the Multiobjective Cloud Autoscaling Problem.

In this section, we present the mathematical formulation of the multiobjective cloud autoscaling problem previously described, which was introduced in [12].

As was previously detailed, each autoscaling stage starts every a predefined period of time throughout the PSE application's execution. Thus, it is necessary to solve the multiobjective cloud autoscaling problem related to each of the autoscaling stages. In this sense, the mathematical formulation introduced in [12] models the multiobjective cloud autoscaling problem to be solved at each autoscaling stage.

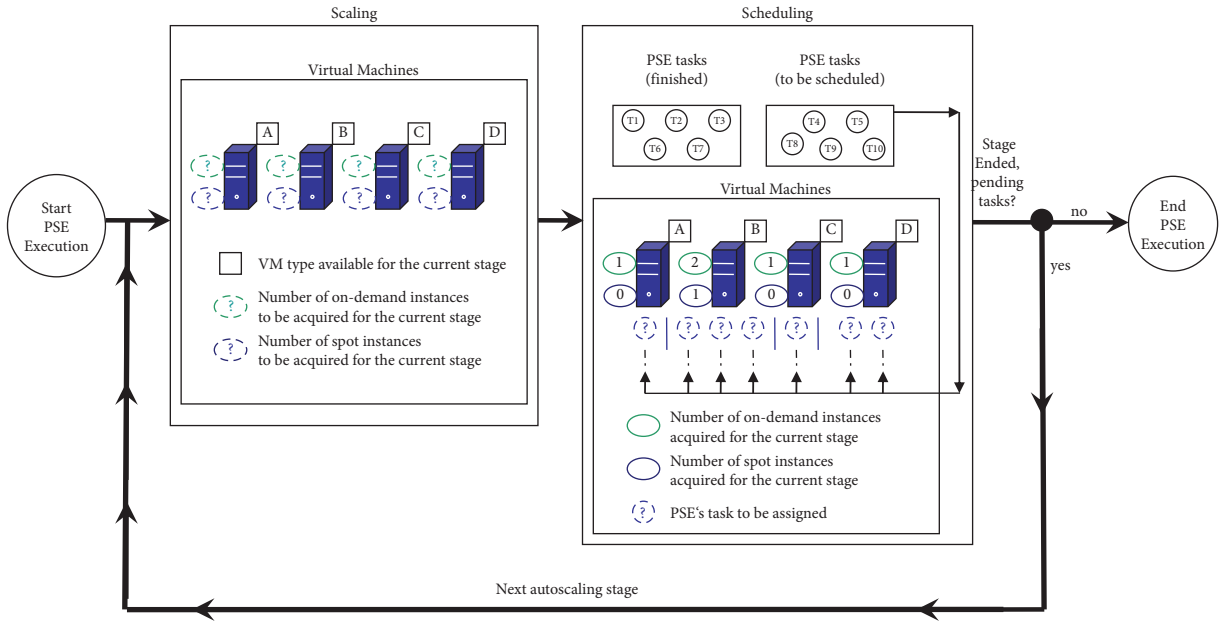


Figure 1: Multiobjective cloud autoscaling problem addressed.

We present below the parameters, decision variables, objectives, and constraints that are utilized in this mathematical formulation to model the multiobjective autoscaling problem to be solved at the beginning of each autoscaling stage. It is worth mentioning that these parameters, decision variables, objectives, and constraints are presented here as they were presented in [12].

2.1.1. Parameters. The parameters considered in each autoscaling stage, and their meaning, are shown in Table 1.

2.1.2. Decision Variables. At the beginning of each autoscaling stage, it is necessary to decide the scaling plan to be applied in the stage. Therefore, the scaling plan is depicted as a tuple X with the following three components: $X = (x^{od}, x^s$, and $x^b)$, where the components x^{od} and x^s contain the decision variables defined to represent the type and number of on-demand and spot instances that will be requested from the CSP for the present stage of autoscaling. In addition, the component x^b contains the decision variables defined to represent the bids to be made to the CSP in order to acquire the desired spot instances. The components of this tuple X, and the decision variables contained in them, are described below in detail.

The first component x^{od} is a vector $(x_1^{od}, x_2^{od}, \ldots x_n^{od})$, where each x_i^{od} is an integer decision variable, which represents the number of on-demand instances of type i that will be requested from the CSP for the present stage of autoscaling. For each variable x_i^{od}, there is a constraint about the range of possible values (Section 2.1.4, Equation (6)).

Then, the second component x^s is a vector $(x_1^s, x_2^s, \ldots, x_n^s)$, where each x_i^s is an integer decision variable, which represents the number of spot instances of type i that will be requested from the CSP for the present stage of autoscaling. For each variable x_i^s, there is a constraint about the range of possible values (Section 2.1.4, Equation (6)).

Finally, the third component x^b is a vector $(x_1^b, x_2^b, \ldots, x_n^b)$, where each x_i^b is an integer decision variable, which represents the bid to be made to the CSP in order to acquire the spot instances of type i for the present stage of autoscaling. For each variable x_i^b, there is a constraint about the range of possible values (Section 2.1.4, Equation (8)).

2.1.3. Optimization Objectives. Given T, which refers to the pending tasks set of the application at the beginning of the present stage of autoscaling, the problem related to this stage involves determining the scaling plan X (i.e., the values for the decision variables in (x^{od}, x^s, x^b)) which minimizes the makespan, the monetary cost, and the spot instance interruptions. This problem is defined by Equation (1), which includes the three optimization objectives considered (Equations (2)–(5)). Besides, this problem is subject to several constraints (Equations (6)–(9)).

$$\min\left(\text{makespan}(X), \text{cost}(X), \text{interruptionImpact}(X)\right). \tag{1}$$

The term *makespan* (X) refers to the estimated computing time of running the tasks in T on the detailed instances in the components x^{od} and x^s of the scaling plan X. In order to estimate such computing time, the well-known scheduling algorithm ECT (earliest completion time) [23] is applied. This algorithm is as follows: for each task t in T, this algorithm first estimates the completion time of task t on each one of the instances detailed in the components x^{od} and x^s of the scaling plan X. After that, this algorithm schedules the task t to the instance that, in principle, might ensure the earliest completion time and finally records the estimated start time and duration of the task t on such an instance.

Table 1: Mathematical formulation parameters.

Parameter	Meaning
T	Set of application's pending tasks at the beginning of the present autoscaling stage
I	Set of VM types available in the cloud environment for the present stage
N	Number of VM types available in the cloud environment for the present stage
$X_i^{\min}$	Minimum number of instances to be requested from the CSP for the VM type i, at the present autoscaling stage
$X_i^{\max}$	Maximum number of instances to be requested from the CSP for the VM type i, at the present autoscaling stage
$Price_i$	Monetary cost of 1 on-demand instance of type i for the duration of the present stage
$S_i^{\text{current Price}}$	Current cost of 1 spot instance of type i for the duration of the present stage
B	Total monetary budget for the present stage

Once the estimated start time and duration of each task in T have been recorded, the algorithm estimates the term *makespan*(X) by applying (2). In the following equation, $ST(t, x^{od}, x^s)$ refers to the start time recorded by the algorithm for the task t, and $d(t, x^{od}, x^s)$ refers to the duration recorded by the algorithm for the task t.

$$\text{makespan}(X) = \max_{t \in T}\left\{ST\left(t, x^{od}, x^s\right) + d\left(t, x^{od}, x^s\right)\right\} - \min_{t \in T}\left\{ST\left(t, x^{od}, x^s\right)\right\}. \tag{2}$$

Then, the term cost(X) refers to the monetary cost of acquiring the instances detailed in the components x^{od} and x^s of the scaling plan X for the duration of the current autoscaling stage. The term cost(X) is determined by (3). In this equation, x_i^{od} refers to the on-demand number of instances of type i depicted in x^{od} and $price_i$ refers to the monetary cost of one on-demand instance of type i for the duration of the current autoscaling stage. Then, x_i^s refers to the spot number of instances of type i depicted in x^s, and x_i^b refers to the bid detailed in x^b for acquiring one spot instance of type i.

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

Finally, the term *interruptionImpact(X)* refers to the possible impact of spot instance interruptions when T's tasks are executed, conforming to the spot number of instances detailed in x^s, and their corresponding bids detailed in x^b. Considering that the interruptions depend on the variation of the monetary cost of the spot instances over time, regarding the bids made by the user for the spot instances, it is not possible to anticipate if, or when, interruptions will happen. Because of this, a function that computes the probability of interruption occurrences, regarding distinct bids, is utilized to define the term *interruptionImpact(X)*. The term *interruptionImpact(X)* is defined by equation (4), where $x_i^s \times v\text{CPU}_i$ is the total number of virtual CPUs of the spot instances of type i, and $P_i(x_i^b)$ refers to the probability of interruption given the bid for the spot instances of type i.

The probability function $P_i(\cdot)$ for the spot instances of type i is computed by considering the number of times an interruption occurs for a predefined number of bid levels though a history of spot prices. This function of probability is defined by equation (5).

In equation (5), $S_{ij} = \left\{s_{ij}^{(1)}, \cdots, s_{ij}^{(m)}\right\}$ is the j^{th} of w series of time stamped spot prices obtained from the historical data for the VM type i (see Section 5.2 for more details about the historical data regarded in our experiments). Later, the equation calculates the number of series in which at least one value is greater than the bid price x_i^b.

$$\text{interruptionImpact}(X) = \sum_{i=1}^{n} x_i^s \times vCPU_i \times P_i\left(x_i^b\right), \tag{4}$$

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

2.1.4. Constraints. Equations (6)–(9) define the constraints considered as part of the cloud autoscaling problem related to the current autoscaling stage.

Equation (6) defines constraints about the number of instances to be required from the CSP for each VM type i, for the present autoscaling stage. In this equation, $X_i^{\min}$ is the minimum possible instances number of type i. In this respect, $X_i^{\min}$ is defined by the instances number of type i that are running at least one task scheduled in an earlier autoscaling stage. In this sense, it is necessary to mention that if an instance is executing tasks scheduled in earlier autoscaling stages, the instance will continue executing these tasks in the present autoscaling stage. The term $X_i^{\max}$is the maximum possible instances number of type i, and is defined by the instances number of type i available in the cloud environment at the present autoscaling stage. This number is determined by the CSP.

$$X_i^{\min} \le x_i^{od} + x_i^s \le X_i^{\max}. \tag{6}$$

Equation (7) defines a constraint about the minimum number of instances indicated in the scaling plan X for the current autoscaling stage. In this respect, at least one instance must be indicated for such a stage. In this equation, considered instances are instances that are running tasks scheduled in preceding autoscaling stages, and new instances to be included in X.

$$\sum_{i=1}^{n} x_i^{od} + x_i^s \ge 1. \tag{7}$$

Equation (8) defines constraints about the bid to be made to the CSP for acquiring the spot instances of each VM type i for the current autoscaling stage. In this equation, $S_i^{\text{current Pr ice}}$

is the current (actual) cost of the spot instances of type i, and the bid x_i^b must be at least $S_i^{\text{current Price}}$. Then, the term $price_i$ is the monetary cost of the on-demand instances of type i, and the bid x_i^b must be at most $price_i$.

$$S_i^{\text{current Price}} \leq x_i^b \leq \text{price}_i. \tag{8}$$

Equation (9) defines a constraint about the total monetary cost of the scaling plan X. This cost must be lower than or equal to a given monetary budget B. This budget might represent, for instance, available monetary credits granted by the CSP to execute applications (AWS credits how to: https://www.parkmycloud.com/blog/aws-credits/) or a threshold imposed by the user on the amount of money to invest.

$$\text{cost}(X) \leq B. \tag{9}$$

3. Multiobjective Cloud Autoscaler MOEA

The multiobjective cloud autoscaler MOEA was introduced recently in [12], for addressing the multiobjective cloud autoscaling problem described in Section 2. Thus, this autoscaler considers that, for executing the tasks of a given PSE application on the VM instances available in a public cloud environment, it is necessary to develop a sequence of different autoscaling stages. Each of these autoscaling stages starts every predefined period of time and implies a different multiobjective autoscaling problem. In this sense, the autoscaler considers that every autoscaling stage starts each hour since the minimum unit of time to acquire a VM instance in Amazon EC2 is one hour.

Because of the factors above mentioned, the autoscaler MOEA develops an iterative process until all the tasks inherent to the PSE application are executed. In this process, each iteration is related to a different autoscaling stage and involves developing three sequential phases for solving the problem related to the autoscaling stage. In the first phase, the autoscaler applies the algorithm NSGA-II to get an approximation of the optimal Pareto set of scaling plans feasible for the stage. In the second phase, the autoscaler selects one scaling plan from the Pareto set provided by the first phase. Finally, in the third phase, the autoscaler applies the selected scaling plan and, after that, schedules the tasks on the VM instances acquired according to such a plan.

The iterative behavior of the autoscaler MOEA is shown in Figure 2, and the phases of each iteration are described below in detail.

3.1. First Phase: Applying a Multiobjective Optimization Algorithm. In this phase, the autoscaler MOEA considers the multiobjective autoscaling problem concerning the present autoscaling stage. Then, the autoscaler applies a multiobjective optimization algorithm to obtain an approximation of the optimal Pareto set for the problem. This means a set of feasible scaling plans with distinct trade-offs among the three optimization objectives considered as part of the problem.

Regarding the algorithm applied by the autoscaler MOEA, the well-known multiobjective evolutionary algorithm NSGA-II [13] is applied. Via this algorithm, a Pareto set of solutions is obtained, where each of the solutions encodes a feasible scaling plan and the solutions have different trade-offs among the optimization objectives considered. In this algorithm, each solution is encoded as described in Section 4.1.1.

It is worth mentioning that in this article, we explore the incorporation of other algorithms to develop this first phase. In this respect, we consider the algorithms E-NSGA-III [18, 19], SMS-EMOA [20], and SMPSO [21]. In addition, we consider the algorithm NSGA-III used in [14] as a reference for comparison purposes. This is because in [14], it has been shown that the performance of the autoscaler MOEA improves considerably when the algorithm NSGA-II is replaced by this algorithm NSGA-III. The main characteristics of the considered algorithms are described in Section 4.

3.2. Second Phase: Selecting the Best Solution. In this phase, MOEA chooses one solution from the Pareto set obtained by the first phase for solving the multiobjective autoscaling problem inherent to the current autoscaling stage.

Concretely, the autoscaler chooses the solution of the Pareto set that is able to minimize the distance to an ideal solution. This means finding a solution that achieves makespan, cost, and interruption probability values equal to 0. To calculate how distant Pareto solutions to the ideal solution are, the autoscaler utilizes the recognized L_2-norm metric. By using this metric, the autoscaler analyzes simultaneously the makespan, monetary cost, and interruption probability of all solutions in the Pareto set and is able to consider the trade-off between the optimization objectives of each solution. To calculate the makespan, monetary cost, and interruption probability of all the solutions of the Pareto set, the autoscaler uses Equations (2)–(4), respectively.

3.3. Third Phase: Acquiring VM Instances and Scheduling Tasks on Them. In this phase, the autoscaler MOEA considers the solution chosen in the second phase, with the aim of determining the virtual infrastructure that will be required from the CSP to execute the tasks in T. Notice that T concerns the application's pending tasks, which were set at the beginning of the present autoscaling stage, as we described in Section 2.1.1.

Concretely, the autoscaler requests the on-demand instance number indicated in the solution for each VM type in I. Besides, the autoscaler requests the spot instance number indicated in the solution for each VM type in I, while detailing the associated bid indicated in the solution for the spot instances. Note that I is the set of available VM types in the cloud, as mentioned in Section 2.1.1.

Once the autoscaler acquires the requested instances from the CSP, it schedules the T's tasks on such instances. To do that, the autoscaler utilizes the scheduling algorithm ECT, which was mentioned in Section 2.1.3.

4. Multiobjective Optimization Algorithms

As detailed in Section 3, the autoscaler MOEA applies a metaheuristic multiobjective optimization algorithm in the

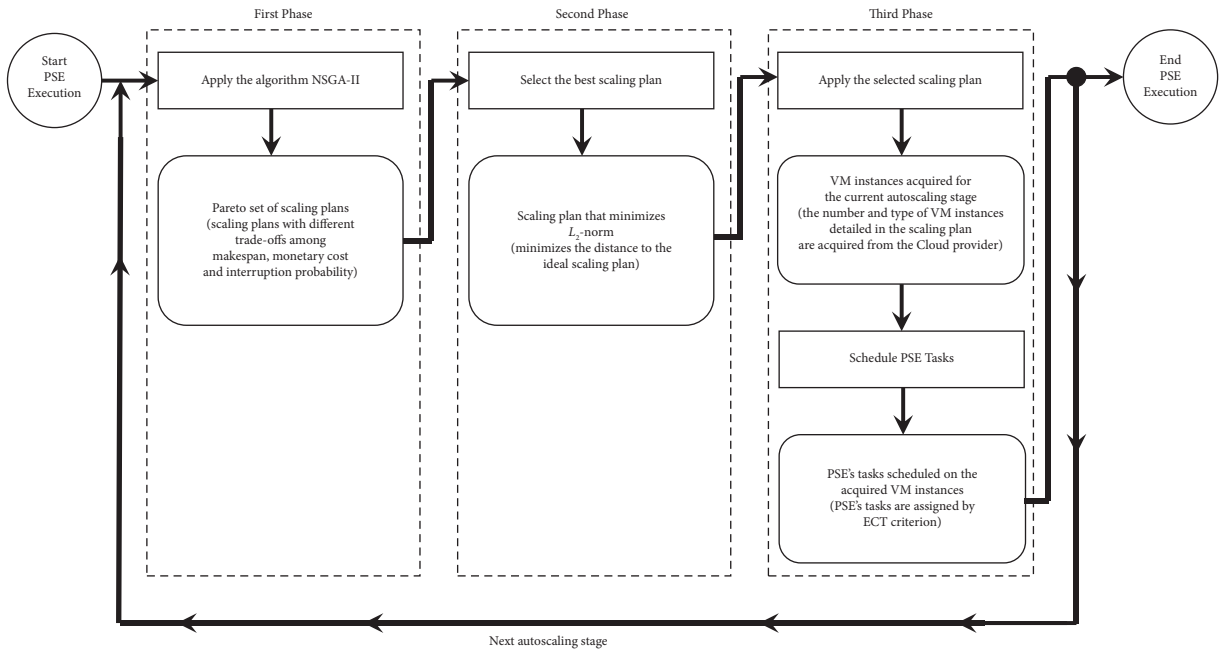


Figure 2: Autoscaler MOEA.

first phase of each iteration with the aim of solving the autoscaling problem related to each iteration, which involves three optimization objectives (i.e., minimizing the makespan, monetary cost, and spot instance interruptions). In this respect, metaheuristic algorithms are aimed at exploring the solution space of a given optimization problem with two or more optimization objectives, possibly in conflict, for an approximation to the optimal Pareto set that includes all the nondominated solutions corresponding to the multiobjective optimization problem. These solutions are characterized by outperforming all the other solutions in the solution space in at least one of the optimization objectives. Thus, the metaheuristics are able to provide a near-optimal Pareto set that contains solutions with very distinct trade-offs among the optimization objectives [24, 25].

In the autoscaler MOEA, the algorithm is utilized for obtaining a Pareto set of candidate scaling plans with very distinct trade-offs among the optimization objectives. Then, the autoscaler selects and applies one scaling plan from the Pareto set obtained by the algorithm. Thus, the better this algorithm is regarding the quality of the Pareto set provided (i.e., diversity, distribution, and convergence to the optimal Pareto set), the better the selected scaling plan could be (i.e., the closer the selected scaling plan could be to the ideal scaling plan), which would impact positively on the performance of the autoscaler in relation to the optimization objectives. In this regard, as was reported in [14], the performance of this autoscaler greatly improves when the algorithm NSGA-II utilized in the first phase is replaced by a more recent and well-known multiobjective evolutionary algorithm named NSGA-III [15]. However, the algorithm NSGA-III has limitations with respect to the diversity of the resulting Pareto set [16, 17, 26], which can negatively impact the performance of the autoscaler. Thus, the incorporation of other algorithms into the first phase of this autoscaler could benefit its performance and, therefore, its applicability to complex real-world PSE applications.

Considering the arguments explained above, we propose analyzing the incorporation of other algorithms into the first phase of the autoscaler MOEA with the aim of enhancing the performance of this autoscaler in relation to the considered optimization objectives. Regarding this, we take into account three known algorithms that have relevant behavioral differences with the algorithm NSGA-III used in [14]. Note that, given the analysis reported in [14], the autoscaler MOEA with this algorithm NSGA-III is considered here as a baseline. One of the three considered algorithms is the multiobjective evolutionary algorithm E-NSGA-III [18, 19]. The other two considered algorithms are the multiobjective evolutionary algorithm SMS-EMOA [20], and the multiobjective particle swarm optimization algorithm SMPSO [21]. The main characteristics of these three algorithms, as well as the main characteristics of the algorithm NSGA-III used in [14], are presented below.

4.1. Algorithm NSGA-III. The algorithm NSGA-III [15] is a variant of the algorithm NSGA-II [13]. NSGA-III is characterized by using a survival selection process that considers a set of reference points with the aim of preserving diversity as well as an even distribution of the Pareto set.

In the NSGA-III-based autoscaler utilized in [14], the first step is generating an initial population with a given number s of solutions. Each one of these solutions depicts a feasible scaling plan and is encoded as described in Section 4.1.1. To generate the s encoded solutions, the random-based process described in Section 4.1.1 is used. Once the initial population is created, the algorithm goes through a number

of iterations until a predefined termination criterion is reached. In each iteration t, the algorithm starts randomly selecting $s/2$ pairs of solutions from the present population, named P_t. Then, the algorithm applies the crossover process SBX (Simulated Binary Crossover) to each of the pairs of solutions, under a crossover probability named P_c, and also under a crossover distribution index named D_c, for generating s new solutions. Then, the algorithm applies the mutation process PM (polynomial mutation) to each of the s new solutions, under a mutation probability named P_m, and a mutation distribution index named D_m. Thus, the algorithm generates an offspring population with s solutions.

After the algorithm generates the offspring population, this population is joined with the current population, obtaining a combined population C with 2^*s solutions. Then, the algorithm chooses s solutions from C in order to generate a new population P_{t+1} for the following iteration. To choose these s solutions from C, the algorithm first calculates the nondomination level of each solution in C and then groups the solutions in C depending on their nondomination levels. These groups are ordered as (G_1, G_2, ...) from the one with the best level to the one with the worst level. Then, to compose the new population P_{t+1}, this algorithm incorporates each group into this population, one at a time, considering the order of the groups, until the P_{t+1} size is equal to s or exceeds s. If the P_{t+1} size is equal to s, the following iteration begins at P_{t+1}. Otherwise, if the P_{t+1} size exceeds s, then the last group G_l incorporated into P_{t+1} is reduced. Concretely, the solutions from group G_1 to group G_{l-1} are incorporated into population P_{t+1}, and then the k solutions remaining are selected from group G_l, where $k = s - |G_1 \cup \ldots \cup G_{l-1}|$.

To choose the k solutions remaining from group G_l, a selection process is used by the algorithm, which considers a set of reference points. This process begins with defining reference points that are set evenly and widely distributed on the normalized hyperplane related to the considered optimization objectives. These objectives are detailed in Section 2.1.3. After that, this process focuses on selecting solutions from G_l that are related to these reference points. Thereby, this process encourages the choice of diverse and evenly distributed solutions, preserving both the diversity and the even distribution of the population P_{t+1}.

In relation to the termination criterion used by the algorithm to stop the iterations, this criterion is achieving a predefined number of evaluations (i.e., a predefined number of generated solutions). When this criterion is achieved, this algorithm supplies the Pareto set inherent to the population of the last iteration as the obtained result.

4.1.1. Encoding of Solutions. The algorithm NSGA-III utilizes the same solution encoding as the algorithm NSGA-II mentioned in Section 3.1. This encoding of solutions is described below.

Each one of the solutions is encoded as a vector with as many positions as $3 \times n$, considering that n is the number of types of available VMs in the cloud environment for the present stage, as detailed in Section 2.1.1. The positions $[1, n]$ of this vector specify the number of on-demand VMs to be purchased for each one of the n types. Such positions have integer values ranging between the minimum possible number of on-demand VMs to be requested and the maximum number of available on-demand VMs for each one of the n types. Subsequently, the positions $[n+1, 2 \times n]$ of this vector specify the number of spot VMs to be purchased for each one of the n types. Such positions have integer values ranging between the minimum possible number of spot VMs to be requested and the maximum number of available spot VMs for each one of the n types. Lastly, the positions $[(2 \times n) + 1, 3 \times n]$ of this vector specify the bid to be made for the spot VMs of each one of the n types. Such positions have real values ranging between the present spot price and the on-demand price for each of the n types. It is worth mentioning that this vector is similar to the tuple $X = (x^{od}, x^s,$ and $x^b)$ described in Section 2.1.2 to represent a scaling plan.

To generate the encoded solutions for the initial population of the algorithm, we used a process based on randomness. In order to create each encoded solution, this process behaves as follows: first, the process considers the number n of VM types and creates an empty vector with $3 \times n$ positions. Then, the process defines the values for the positions $[1, n]$ of this vector. In this respect, for each one of the VM types i ($i = 1, \ldots, n$), this process randomly selects an integer value between the minimum possible number of on-demand instances to be requested and the maximum on-demand VMs number available for the type i and copies the selected value in the position i of this vector. Then, the process defines the values for the positions $[n+1, 2 \times n]$ of this vector. In this sense, for each one of the VM types i, this process randomly selects an integer value between the minimum spot VM number to be requested and the maximum spot VMs number available for the type i, and copies the selected value in the position $(n + i)$ of this vector. Finally, the process defines the values for the positions $[(2 \times n) + 1, 3 \times n]$ of this vector. Specifically, for each one of the VM types i, this process randomly selects a real value between the present price of the spot VM and the price of the on-demand VM for the type i, and copies the selected real value in the position $((2 \times n) + i)$ of this vector.

Once the values of all the positions of the vector are defined, the process develops an additional analysis in order to decide if these values are accepted or must be redefined. Specifically, the process analyzes the total number of instances to be acquired (i.e., the sum of the values in the positions $[1, 2 \times n]$), to guarantee that at least 1 instance will be acquired. Moreover, the process analyzes the total cost of the instances to be acquired to guarantee that the total cost will be lower than or equal to the total monetary budget for the current stage. When the total number of instances to be acquired is greater than or equal to 1, and the total cost is less than or equal to the total monetary budget, the values defined for the positions are accepted. Otherwise, if the total number of instances to be acquired is less than 1, or if the total cost is higher than the total monetary budget, the process defines new possible values for the positions of the vector, and after that, it performs the additional analysis

previously described. In this way, the process generates feasible encoded solutions for the initial population of the algorithm.

4.2. Algorithm E-NSGA-III. E-NSGA-III [18, 19] is a recent extension of the algorithm NSGA-III and has been presented in the literature with the aim of improving the diversity, distribution, and convergence of the Pareto sets generated by NSGA-III. E-NSGA-III is characterized by including a number of extreme solutions within the initial population, for enhancing the diversity and well-distribution of the Pareto set.

The general behavior of the algorithm E-NSGA-III is similar to that of NSGA-III. In the two algorithms, the first step is generating an initial population with a given number s of solutions. Each of these solutions encodes a feasible scaling plan and is encoded as described in Section 4.1.1. To generate the s encoded solutions, the random-based process described in Section 4.1.1 is utilized. After that, in each of the iterations, these two algorithms use sequentially the crossover process SBX and mutation process PM on pairs of solutions randomly chosen from the current population, to create an offspring population containing s solutions. Then, both algorithms combine the current population and offspring population and apply the same selection process for determining which solutions from this combined population will constitute the population for the following iteration. These algorithms also utilize the same termination criterion to stop their iterations. Nevertheless, these algorithms are different with respect to the generation of the initial population.

In NSGA-III, the initial population is randomly generated by using the random-based process described in Section 4.1.1. Unlike this, the initial population in E-NSGA-III is generated as follows: first, a random population is generated using the random-based process previously mentioned. Then, a number of extreme solutions are included in this randomly generated population. An extreme solution refers to a solution having an optimal value regarding one of the optimization objectives considered, regardless of the values corresponding to the other optimization objectives considered. E-NSGA-III incorporates one extreme solution per optimization objective considered. The incorporation of these extreme solutions to the random population has been proposed for guiding the algorithm to generate more diverse and better distributed nondominated solutions, and thus obtain Pareto sets with better diversity and distribution [18, 19].

4.2.1. Extreme Solutions Defined. As described in Section 2.1.3, three optimization objectives are considered here. Thus, we have included three extreme solutions within the initial population of E-NSGA-III. Specifically, we have included one extreme solution regarding the minimization of the makespan, one extreme solution regarding the minimization of the monetary cost, and one extreme solution regarding the minimization of the interruption probability.

The extreme solution defined with respect to the minimization of the makespan has an optimal makespan. This solution proposes acquiring the maximum number of on-demand instances allowed for the VM type with the highest processing capacity. When the instances detailed in this solution are considered, each of the tasks in T (i.e., tasks to be executed) is assigned to a different on-demand instance of the mentioned type. Thus, an optimal makespan is obtained.

Figure 3(b) shows the extreme solution regarding the minimization of the makespan for the example case presented in Figure 3(a). In this case, VM type 2 has the highest processing capacity. Thus, the extreme solution proposes to acquire the highest on-demand instance number permitted for VM type 2 (i.e., 10 on-demand instances). Then, the 8 tasks in the set T are scheduled on the on-demand instances proposed by the solution via the algorithm ECT. As described in Section 2.1.3, the algorithm ECT assigns each task to the instance that, in principle, might ensure the earliest completion time. Thus, each of the 8 tasks in T is assigned to a distinct on-demand instance of VM type 2, guaranteeing the optimal makespan.

The extreme solution defined in respect of the minimization of the cost has an optimal monetary cost. This solution proposes to acquire only one spot instance of the VM type at the lowest monetary cost for spot instances. Besides, the bid proposed by the solution for acquiring this spot instance is equal to the minimum allowed bid for the spot instances of the mentioned type (i.e., the monetary cost of the spot instances of the mentioned type). Note that on-demand instances are not considered in building this extreme solution since these are more expensive than spot instances for each VM type.

Figure 3(c) shows the extreme solution regarding the minimization of the monetary cost for the example case presented in Figure 3(a). In this case, VM type 1 has the lowest monetary cost for spot instances. Thus, the extreme solution proposes to acquire one spot instance of the VM type 1, and also proposes the minimum bid allowed to acquire this spot instance (i.e., the monetary cost of one spot instance of the VM type 1). When the spot instance proposed by the extreme solution is considered, the tasks within the set T are sequentially scheduled on this instance.

The extreme solution defined with respect to the minimization of the interruption probability has an optimal interruption probability. This solution proposes acquiring only on-demand instances for all the VM types. Specifically, this solution proposes a possible number of on-demand instances for each of the VM types. In addition, this solution proposes not to acquire spot instances for any of the VM types. Recall that, in contrast to the spot instances, the on-demand instances are not subject to interruptions. Thus, this solution has an optimal interruption probability.

Figure 3(d) shows the extreme solution regarding the minimization of the interruption probability for the example case presented in Figure 3(a). For each of the four VM types in this case, the extreme solution proposes a possible number of on-demand instances. Specifically, for each of the VM types, the number of on-demand instances proposed by the solution is higher than (or equal to) the minimum number of

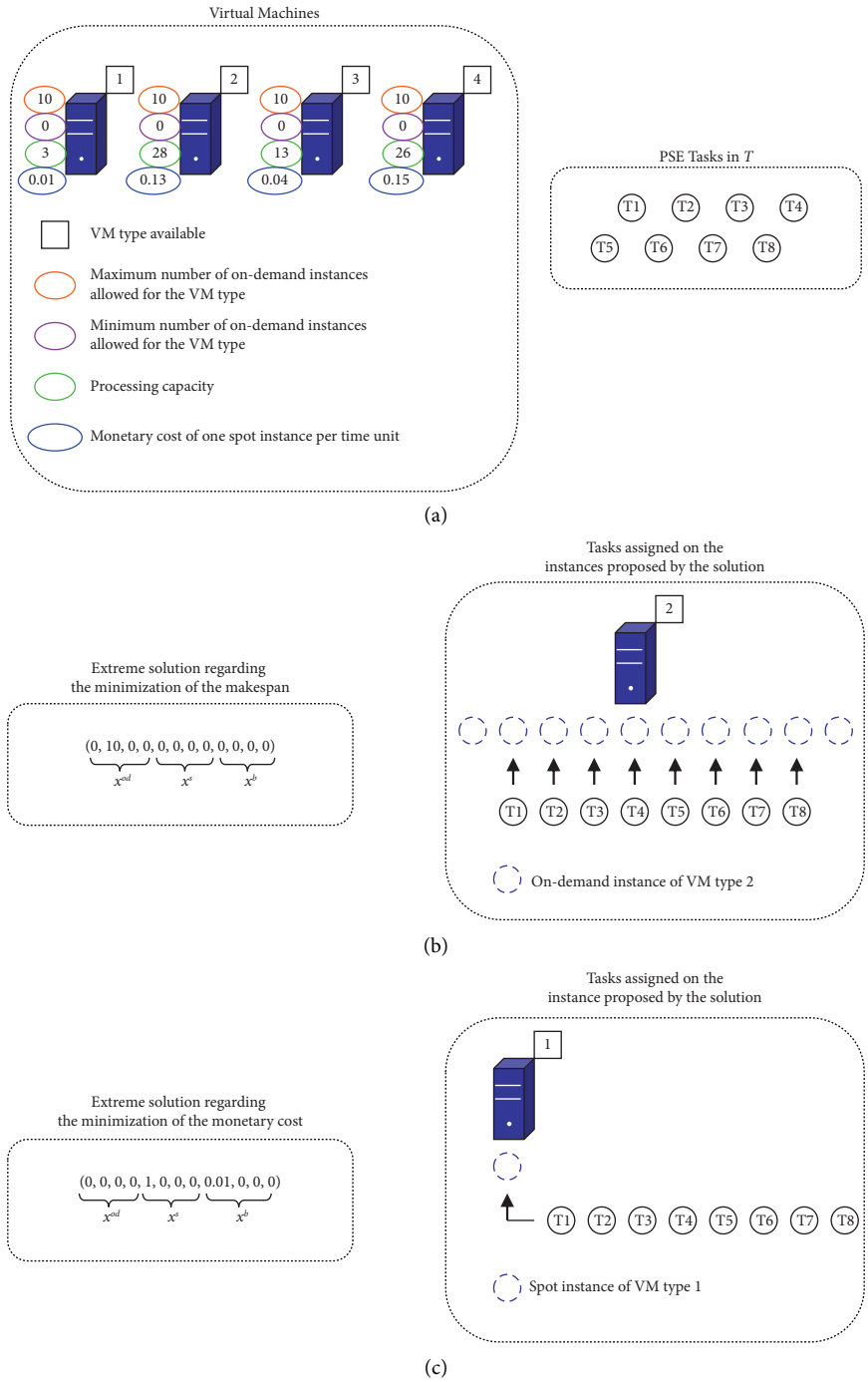
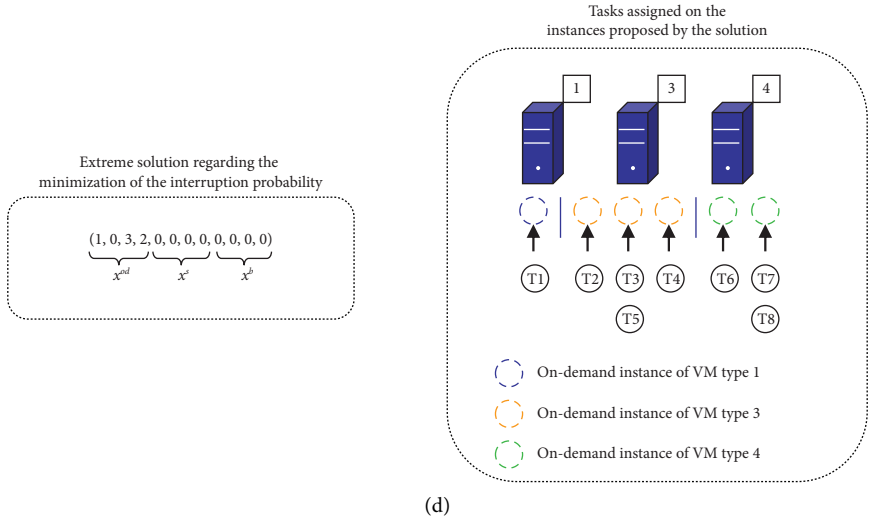


FIGURE 3: Continued.



(d)

Figure 3: Extreme solutions for an example case.

on-demand instances permitted and lower than (or equal to) the maximal number of on-demand instances permitted. For instance, for the VM type 1, the solution proposes 1 on-demand instance, which is higher than 0 (i.e., the minimum number of on-demand instances permitted for the VM type 1) and lower than 10 (i.e., the maximal number of on-demand instances permitted for the VM type 1). Figure 3(d) illustrates a schedule of the tasks within the set T for the instances proposed by the solution.

4.3. Algorithm SMS-EMOA. SMS-EMOA [20] uses a survival selection process led by the nondominated sorting combined with the hypervolume metric, for preserving diversity as well as distribution of the Pareto set.

In the first step, SMS-EMOA creates a random initial population with a given number s of solutions. Each of the solutions encodes a feasible scaling plan, encoded as described in Section 4.1.1. To generate the s encoded solutions, the random-based process described in Section 4.1.1 is used.

After the algorithm generates the initial population, it follows a number of iterations until the termination criterion is achieved. In each iteration t, this algorithm applies the operator SBX and the operator PM to a pair of solutions chosen randomly from the current population named P_t, which generates a new solution. After that, the algorithm applies the selection process to determine if the new solution will be included in the next population P_{t+1} for the following iteration. In this respect, if the newly-created solution improves the hypervolume, it is added to the next population.

The selection process begins by joining the s solutions in P_t with the newly-created solution. After that, the process analyzes the nondomination level of each one of these $s+1$ solutions, and groups these solutions according to their nondomination levels. Then, these groups are sorted as $\{G_1, \cdots G_2, \cdots G_v\}$, from the one with the best level to the one with the worst level. Then, the process takes one solution from the group with the worst level G_v, with the aim of obtaining the s solutions that will constitute the population P_{t+1}. Specifically, the process takes out the solution x_i that belongs to G_v and minimizes equation (10), considering $i = 1, \ldots, |G_v|$.

In equation (10), $S(G_v)$ refers to the hypervolume of G_v, and $S(G_v - \{x_i\})$ refers to the hypervolume of $(G_v - \{x_i\})$. Then, the value of $\Delta_S(x_i, G_v)$ measures the contribution of solution x_i to the hypervolume of its group G_v. Therefore, the selection process considers the contribution of each of the solutions in G_v to the hypervolume and then maintains the solutions that maximize it.

In relation to the termination criterion utilized by the algorithm to stop the iterations, this algorithm uses the same criterion as the algorithm NSGA-III. Once this criterion is achieved, this algorithm supplies the Pareto set corresponding to the population of the last iteration, as the result obtained.

$$\Delta_S(x_i, G_v) = S(G_v) - S(G_v - \{x_i\}). \tag{10}$$

4.4. Algorithm SMPSO. SMPSO [21] is an extension of the known OMOPSO algorithm. SMPSO is characterized by using a process aimed at constraining the velocity factor utilized for updating the solutions of the population, in order to preserve diversity and the distribution of the solutions.

In this algorithm, the first step implies creating a random initial population with a given number s of solutions. Each of the solutions encodes a feasible scaling plan, encoded as described in Section 4.1.1. To generate the s encoded solutions, the random-based process described in Section 4.1.1 is used. Besides, each solution has an initial velocity factor associated, and also an initial memory associated. The memory is used to store the updates to the solution over time. The second step implies creating a leader archive that initially contains all nondominated solutions belonging to the initial population.

Once the algorithm generates the initial population and also the leaders archive, the algorithm develops a number of

iterations until the termination criterion is achieved. In each iteration t, this algorithm starts updating the velocity factor of each one of the s solutions into the current population named P_t. In this sense, the current velocity factor of each solution j ($j = 1,\ldots,s$) is updated considering the distance between j and the best solution in the memory of j and also the distance between j and the best solution reached by the algorithm until the iteration t. Then, the algorithm applies a constriction process on the velocity factor of each solution j to avoid very high/low values for this velocity factor and thus avoid upper/lower bound values for the variables of the solution, which favors the generation of diverse solutions. Subsequently, the algorithm updates each solution j considering the constrained velocity factor of the solution. In this respect, the algorithm updates the values of the variables of each solution j by adding the constrained velocity factor of the solution to the current values of the variables. For a detailed description of the equations utilized by this algorithm to update and constraint the velocity factor and update each solution, we refer to [21].

Then, the algorithm applies the mutation operator PM on each updated solution j under a given mutation probability named P_m and under a given mutation distribution index named D_m. After that, the algorithm evaluates each of the obtained solutions and updates the memory of each of the solutions for the following iteration. Moreover, this algorithm updates the leader archive for the next iteration with the aim of preserving the nondominated solutions generated throughout the search process.

The termination criterion used by the algorithm to stop the iterations is the same as NSGA-III. After this criterion is achieved, the leaders archive of the last iteration is returned by the algorithm.

5. Computational Experiments

To comparatively analyze the performance of the autoscaler MOEA with each of the four analyzed algorithms, NSGA-III, E-NSGA-III, SMS-EMOA, and SMPSO, we carried out extensive computational experiments via simulation, which are presented in this section.

First, we describe the four real-world PSEs used in the experiments. Then, we present the different VM types utilized (on-demand and spot) and their specifications. After that, the experimental setting considered for carrying out the experiments is detailed. Finally, we report and analyze in detail the results achieved by these experiments.

5.1. PSE Applications. Next, the subsections describe four real-world applications from the molecular dynamics and meteorology areas. We also describe how we derived, from these applications, the PSE tasks considered in the simulated experiments.

5.1.1. Melting Process of Gold Nano-Clusters (MPG). Melting process of gold nano-clusters: studies the thermodynamics of the melting transition in Au (gold) nanoclusters with 1,985 to 180,313 atoms as spheres of different radius centered at the origin of the face-centered cubic crystal structure (FCC) lattice [27]. This study contributes to the understanding of the melting process of gold, a material that has been used in different technological and biomedical applications [28]. The executions were performed using the Molecular Dynamics software LAMMPS [29] with the embedded atom model (EAM) interaction potential, used in other works as well [30]. The only parameter that changes in the parametric study is the radius of the spheres. Each PSE task has one sphere of Au atoms with a given radius, with a total of 1,985 atoms for the smallest sphere and 180,313 for the biggest sphere. Between all the executions, a total of 3,473 tasks were generated. Figure 4 shows snapshots of the melting process with 1,985 atoms, colored by their coordination number. This analysis represents the number of neighbors atoms any particular atom has in a given radius around it. With a radius of 3.3 Lennard Jones units, red atoms have 12 neighbors and blue atoms have 5 neighbors.

The study focuses on analyzing where the transition takes place, or the "melting step," by identifying and describing the amounts of the following two types of atoms: SPL (solid-phase-like) and LPL (liquid-phase-like). The energy and entropy change in this melting step were set as functions of the number of atoms. This energy change with temperature allows us to quantify the amount of atoms in SPL and the transition to LPL atoms and to model the melting step in each size of nano-clusters.

5.1.2. Granular Mechanics Simulations (GMS). Granular mechanics studies the behavior of aggregates of silica (SiO_2) grains during collisions. The study allows for the analysis of complex properties of dust interactions with relevance in astrophysics for planet formation, cometary comae, and debris discs [31]. The simulation consisted of a projectile aggregate striking a larger immobile target aggregate both are formed by grains of silica. A parameter sweep was performed by modifying several initial conditions, resulting in 50 different tasks (not all combinations of parameters were executed). The modified parameters were eight impact velocities (5, 2.5, 1, 0.75, 0.5, 0.25, 0.1, and 0.05 m/s), three filling factors (0.15, 0.25, and 0.35), and three different sizes of projectile and target (small, med and big) [32]. The combination of the three different sizes of projectile and target, with the three filling factors, produces different amounts of grain with each combination.

(i) Size small with a 0.15 filling factor has 11,200 grains
(ii) Size small with a 0.25 filling factor has 18,785 grains
(iii) Size small with a 0.35 filling factor has 26,206 grains
(iv) Size med with a 0.15 filling factor has 31,087 grains
(v) Size med with a 0.25 filling factor has 51,910 grains
(vi) Size med with a 0.35 filling factor has 72,353 grains
(vii) Size big with a 0.15 filling factor has 51,888 grains
(viii) Size big with a 0.25 filling factor has 86,734 grains

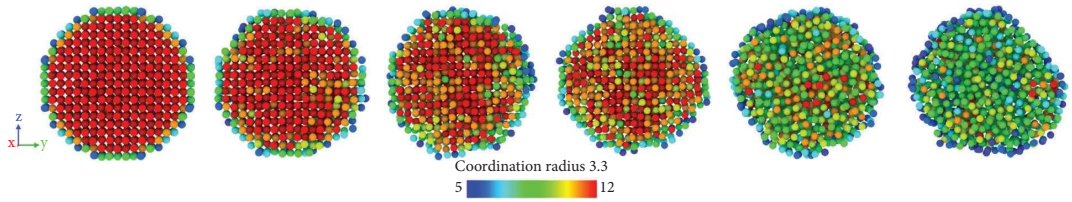


Figure 4: Melting process of Au nano-clusters.

(ix) Size big with a 0.35 filling factor has 120,894 grains

The tasks were executed using the molecular dynamics software LAMMPS [29]. Figure 5 shows a slice of the center of the sample in five moments of the collision study, the color scale represents the velocity in the Z axis of the projectile and the target. This simulation employs a fill factor of 0.35, and the initial velocity of the projectile is m/s.

5.1.3. Frost Prediction Application (FPA). Predicting frost is a topic of special interest to mitigate damage in different parts of the world [33]. Because frost phenomena occur every year, farmers provide their farms with heathers, sprinklers, and wind turbines as defense methods to minimize crop damage. In turn, these methods are set in operation alarm systems, which perform on-field data acquisition through the use of weather stations, thermometers, and Wireless Sensor Networks (WSNs) [34]. Particularly, WSNs are composed of low-cost devices called sensor nodes, and thus larger areas can be covered. This advantage is important for the study of frosts because depending on the terrain characteristics (e.g., the existence of vegetation and proximity to mountains), the phenomenon could or could not occur. It has been observed in the farms that depending on the characteristics of the land, the phenomenon occurred in some hectares and not in others.

For performing a frost prediction, the FPA implements the Snyder and Melo–Abreu method [35] on data from a WSN and weather stations (temperature and humidity) and computes dew points on those days that radiation frosts occurred. Besides, temperature, humidity and dew points must have been registered 2 hours after sunset on the prediction day. In this work, for simulating a frost prediction, temperature, and humidity data have been sensed by WSNs and weather stations instrumented in some fields of the Province of Mendoza, Argentina. Concretely, 40 farms were instrumented, each with 10 to 1,000 sensor nodes, depending on the farm's size. Historical temperature values, humidity values, and dew points of 50 days where frosts occurred were considered to make the prediction in each of the farms. The 50-day historical data must match the same month in which the prediction is performed. Historical data was provided by the National Oceanic and Atmospheric Administration (https://www.noaa.gov/). Therefore, the FPA was run with data from different sensor nodes, generating 40 FPA tasks.

5.1.4. Weather Research and Forecasting (WRF). An atmospheric system is described by a weather prediction model [36] through mathematical equations that model physical conservation laws. The model is configured in a computational 3D mesh-grid consisting of several thousand horizontal and vertical direction points. The higher the model resolution, the denser the computational grid spacing. According to the Earth's coverage, weather models can be classified as global or regional.

Then, Weather Research and Forecasting (WRF) [37] is a mesoscale numerical weather prediction application used for researching atmospheric and operational forecasting. The system is useful for many large-scale meteorological applications. WRF performs simulations based on atmospheric conditions.

Forecast computation starts with the known boundary conditions on the Earth's surface at the atmosphere's upper boundary and an initial state based on observations of the current weather. Then, the equations are computed for each time step at each point of the grid of the 3D model until the forecast is completed. Therefore, for obtaining a forecast, the atmospheric data are obtained in real time from a server that holds the data of the last 15 days and that comes from different meteorological stations. The downloaded data is preprocessed to filter the domain area of interest where to perform the forecast, and moreover, static information on the specific soil conditions for that interest zone (grasslands, mountainous areas, etc.) is added. Once the atmospheric data is preprocessed, the forecasting is performed through the WRF model. Solving the WRF equations that compose the model requires large computing capabilities, therefore, it is necessary to have powerful servers.

The execution times have been measured by performing a three-day forecast for central Argentina. The parent domain (highest resolution domain) where the data was downloaded has grid cells with a grid spacing of 5 km (1 km) consisting of 105 (151) grid-points in the west-east direction and 151 (171) grid-points in the north-south direction. The WRF model was run in parallel, and a total of 50 WRF tasks were generated to perform the previously described prediction.

For evaluating the performance of the autoscaler MOEA against four algorithms, different instances of the obtained real base task set comprising 50, 40, and 50 tasks for the GMS, FPA, and WRF were executed to obtain more tasks, i.e., 30, 100, and 300 tasks. In the particular case of MPG,

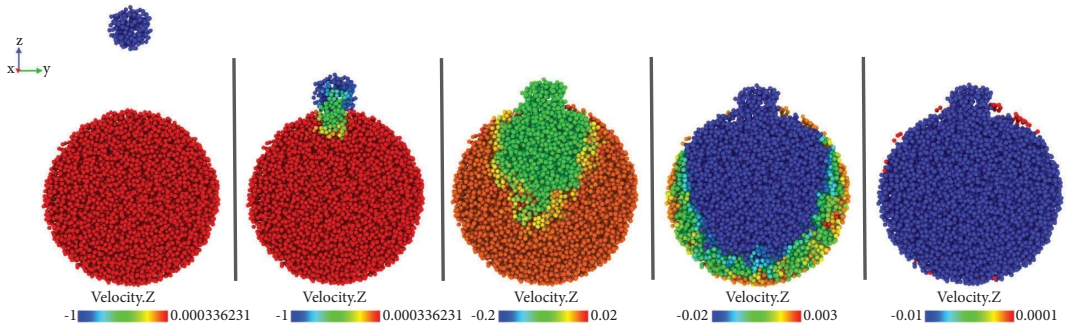


Figure 5: Collision of aggregates of silica (SiO_2).

which is composed of a greater number of tasks, we have restricted the number to 30, 100, and 300 tasks as with the other applications.

5.2. VM Types. During the experiments, we considered the on-demand instance specifications shown in Table 2. The first column presents the different instance types considered. The instance characteristics were set up as the real Amazon Elastic Compute Cloud (EC2) instances. Then, columns 2, 3, and 4 show the number of virtual CPUs (vCPU) available for each of the instance types, the relative computing power of the instance considering all its virtual CPUs (ECU^{tot}), and the relative performance of each of the CPUs (ECU), respectively. Finally, the last column shows the price in US dollars (USD) per hour of computation. Instance-type-wise, we aimed at providing different price and performance configurations.

On the other hand, for the spot instances, we used the history of Amazon EC2 spot prices for the US-west region (Oregon) considered in [12, 14]. The period corresponds to the months between March 7th and June 7th of 2016. The interruption probabilities were computed using data from the first two months. Specifically, we computed how many times a 1-hour sliding window showed spot prices greater than the bid values. Then, the data pertaining to June 2016 was kept for the experiments presented in Subsection 5.4 as the spot price variations over the course of the simulation. The use of the data in such a way allows us to evaluate MOEA, ignoring completely the future evolution of spot prices, as occurs in practice.

5.3. Experimental Setting. We considered the autoscaler MOEA with the algorithm NSGA-III [14] as a reference for comparison purposes, as mentioned in Section 4. Then, we incorporated the algorithms E-NSGA-III, SMS-EMOA, and SMPSO into the autoscaler MOEA. Thus, we obtained four variants of this autoscaler, which differ regarding the multiobjective optimization algorithm used. For simplicity, these variants will be referred to as being detailed in Table 3.

We run each of the four variants of the autoscaler MOEA on each of the applications and sizes presented in Section 5.1, considering the utilization of on-demand and spot instances of the VM types presented in Section 5.2. Considering that these variants are based on nondeterministic algorithms,

Table 2: On-demand instance specifications corresponding to instances used in the evaluation.

VM type	vCPU	ECU^{tot}	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

Table 3: Variants of the autoscaler MOEA.

Variant of the autoscaler MOEA	Name of the variant
MOEA with NSGA-III	MOEA-NSGA-III
MOEA with E-NSGA-III	MOEA-E-NSGA-III
MOEA with SMS-EMOA	MOEA-SMS-EMOA
MOEA with SMPSO	MOEA-SMPSO

each variant was run several times (i.e., 30 times) on each of the applications and sizes to obtain reliable statistical results. For each of the runs, we record the value corresponding to each of the three optimization objectives considered as part of the multiobjective cloud autoscaling problem.

The runs of the four variants of the applications were developed using the well-known CloudSim simulator [22]. CloudSim is one of the simulators usually used in the literature in order to develop computational experiments related to scheduling and resource assignment problems in cloud environments.

To run the variant MOEA-NSGA-III, we utilized the parameter setting suggested in [15] for the algorithm NSGA-III. Moreover, we established *the number of evaluations* of the algorithm NSGA-III since there is no generic suggestion for this parameter in [15]. Note that this parameter refers to the termination criterion utilized by the algorithm to stop its execution. Specifically, the algorithm will stop its execution once the given number of evaluations (i.e., generated solutions) is reached. The parameter setting used for NSGA-III is detailed in Table 4.

Given that the variant MOEA-NSGA-III is considered a reference for comparison purposes, and that it is necessary to guarantee a fair comparison of the four variants, to run the variants MOEA-E-NSGA-III, MOEA-SMS-EMOA, and MOEA-SMPSO, the parameters of the algorithms

Table 4: Configuration for NSGA-III and E-NSGA-III

Parameter	Value
Population size	92
Number of reference points	91
Number of evaluations	36800
P_c	1
D_c	30
P_m	1/15 (the length of each encoded solution is 15)
D_m	20

E-NSGA-III, SMS-EMOA, and SMPSO were set with the same values used for NSGA-III. It is necessary to mention that E-NSGA-III has the same parameters as NSGA-III. Thus, the parameter setting used for E-NSGA-III is detailed in Table 4. In the case of SMS-EMOA, this algorithm does not use reference points like NSGA-III, and so does not have the parameter *number of reference points* utilized by NSGA-III. However, SMS-EMOA utilizes the same crossover and mutation operators as NSGA-III. In the case of SMPSO, this algorithm does not utilize reference points like NSGA-III, and besides, does not use a crossover operator like NSGA-III. Thus, SMPSO does not have the parameters *number of reference points*, P_c, and D_c used by NSGA-III. However, SMPSO uses the same mutation operator as NSGA-III. In addition, this algorithm uses the parameter named *leaders archive size*, which was set to the value of the parameter *Population size*, as suggested in [21]. The parameter settings utilized for SMS-EMOA and SMPSO are detailed in Tables 5 and 6.

Table 5: Parameter setting of SMS-EMOA.

Parameter	Value
Population size	92
Number of evaluations	36800
P_c	1
D_c	30
P_m	1/15
D_m	20

Table 6: Parameter setting of SMPSO.

Parameter	Value
Population size	92
Number of evaluations	36800
Leaders archive size	92
P_m	1/15
D_m	20

5.4. Experimental Results. In Tables 7 and 8, we show the obtained results from the performed experiments regarding the three optimization objectives considered as part of the cloud autoscaling problem addressed. In these tables, column 1 presents the names of the PSE applications used in these experiments. Column 2 indicates the sizes considered for the applications in these experiments, where size refers to the number of tasks in the application, considering one task per parameter setting of the application. Column 3 mentions the four variants of the autoscaler MOEA, which were evaluated in these experiments. Recall that each variant was run several times (i.e., 30 times) on each application and size. Columns 4 and 5 show the average makespan and monetary cost obtained by each variant for each application and size. Column 6 details the average number of task failures obtained by each variant for each application and size, where task failures are associated with spot instance interruptions. Later, column 7 shows the average (Euclidean) L_2-norm metric which trades off makespan, monetary cost, and the number of task failures.

To calculate the average makespan obtained by each variant for each application and size, we have added the 30 makespan values provided by the 30 runs of each variant for each application and size, and then we have divided the sum result by the number of runs (i.e., 30). To calculate the average monetary cost/average number of task failures obtained by each variant for each application and size, we followed a similar procedure, but we considered the 30 monetary cost values/30 values of task failures provided by the 30 runs of each variant for each application and size. Similarly, in order to calculate the average value for each variant with respect to each application and size for the L_2-norm metric, we considered the 30 values provided by the 30 runs of each variant on each application and the size for this metric.

In addition, in Tables 9 and 10, we present the values obtained for the following metrics: average makespan RPD (relative percentage difference), average cost RPD, and average number of task failures RPD, all three computed with respect to MOEA-NSGA-III. Next, we describe these three metrics.

The average makespan RPD metric is the % difference of the average makespan of MOEA-E-NSGA-III (or MOEA-SMPSO, or MOEA-SMS-EMOA) regarding the average makespan of MOEA-NSGA-III, as calculated by the formula $((m^t - m)/m^t) * 100$, where m^t is the average makespan of MOEA-NSGA-III and m is the average makespan of MOEA-E-NSGA-III (or MOEA-SMPSO, or MOEA-SMS-EMOA). If the difference is greater than zero, this means that MOEA-E-NSGA-III (or MOEA-SMPSO, or MOEA-SMS-EMOA) has reached a decrease in average makespan w.r.t. MOEA-NSGA-III. If the difference is below zero, this means that MOEA-E-NSGA-III (or MOEA-SMPSO, or MOEA-SMS-EMOA) has achieved an increase in average makespan w.r.t. MOEA-NSGA-III.

Table 7: Average makespan, average monetary cost, average number of task failures, and average L_2-norm obtained by each variant of the autoscaler MOEA for applications MPG and GMS. For all metrics, lower values are better values. For each application and size, bold values are better than those obtained by *MOEA-NSGA-III*.

Application	Size	Autoscaler	Makespan	Cost	Task failures	L_2-norm
MPG	30	*MOEA-NSGA-III*	13162.40	1.71	0.10	0.33
		MOEA-SMPSO	20809.32	**1.57**	0.20	0.74
		MOEA-SMS-EMOA	**12944.68**	2.74	*__0.00__	0.57
		MOEA-E-NSGA-III	*__12838.21__	*__1.52__	*__0.00__	*__0.26__
	100	*MOEA-NSGA-III*	14971.13	4.44	*0.00	0.16
		MOEA-SMPSO	23190.13	10.46	1.40	0.73
		MOEA-SMS-EMOA	23575.42	12.23	*0.00	0.77
		MOEA-E-NSGA-III	*__13339.48__	*__3.78__	*0.00	*__0.06__
	300	*MOEA-NSGA-III*	38160.13	37.45	0.60	0.67
		MOEA-SMPSO	*__31377.70__	*__30.23__	1.77	**0.50**
		MOEA-SMS-EMOA	47792.46	63.50	**0.57**	0.88
		MOEA-E-NSGA-III	**31839.27**	**30.64**	*__0.00__	*__0.46__
GMS	30	*MOEA-NSGA-III*	1074149.61	531.36	4.38	0.46
		MOEA-SMPSO	1422601.27	**456.40**	**4.29**	0.49
		MOEA-SMS-EMOA	*__1033579.47__	680.52	5.43	0.62
		MOEA-E-NSGA-III	**1050575.97**	*__406.47__	*__2.77__	*__0.30__
	100	*MOEA-NSGA-III*	1021845.42	1731.66	13.30	0.51
		MOEA-SMPSO	1382518.24	*__1612.51__	21.44	0.72
		MOEA-SMS-EMOA	1079659.30	2287.06	**6.41**	0.58
		MOEA-E-NSGA-III	*__915882.64__	**1635.15**	*__6.07__	*__0.38__
	300	*MOEA-NSGA-III*	1021592.20	5461.36	22.18	0.30
		MOEA-SMPSO	1353654.86	**5080.45**	67.25	0.52
		MOEA-SMS-EMOA	1355956.14	7660.57	42.00	0.56
		MOEA-E-NSGA-III	*__918339.45__	*__4613.51__	*__14.64__	*__0.19__

* indicate the best value. The baseline is in italics.

Table 8: Average makespan, average monetary cost, average number of task failures, and average L_2-norm obtained by each variant of the autoscaler MOEA for applications FPA and WRF. For all metrics, lower values are better values. For each application and size, bold values are better than those obtained by *MOEA-NSGA-III*.

Application	Size	Autoscaler	Makespan	Cost	Task failures	L_2-norm
FPA	30	*MOEA-NSGA-III*	6925.12	0.43	*0.00	0.06
		MOEA-SMPSO	9742.57	0.97	0.13	0.52
		MOEA-SMS-EMOA	**6869.20**	2.06	*0.00	0.44
		MOEA-E-NSGA-III	*__6862.83__	*__0.40__	*0.00	*__0.04__
	100	*MOEA-NSGA-III*	10406.34	2.13	*0.00	0.17
		MOEA-SMPSO	17386.56	3.38	0.53	0.78
		MOEA-SMS-EMOA	*__10153.26__	3.35	*0.00	0.29
		MOEA-E-NSGA-III	**10221.36**	*__1.79__	*0.00	*__0.13__
	300	*MOEA-NSGA-III*	12483.92	6.98	*0.00	0.24
		MOEA-SMPSO	16138.69	10.03	1.60	0.62
		MOEA-SMS-EMOA	16684.92	13.50	*0.00	0.71
		MOEA-E-NSGA-III	*__11972.73__	*__6.34__	*0.00	*__0.19__
WRF	30	*MOEA-NSGA-III*	60741.41	11.33	*0.00	0.93
		MOEA-SMPSO	61388.53	**9.98**	3.07	0.96
		MOEA-SMS-EMOA	62653.31	*__6.79__	*0.00	0.92
		MOEA-E-NSGA-III	*__56088.10__	**10.02**	*0.00	*__0.81__
	100	*MOEA-NSGA-III*	60180.73	47.02	0.40	0.86
		MOEA-SMPSO	63291.46	55.04	4.57	0.97
		MOEA-SMS-EMOA	63530.41	*__24.51__	3.33	0.88
		MOEA-E-NSGA-III	*__58293.80__	**28.53**	*__0.00__	*__0.76__
	300	*MOEA-NSGA-III*	59934.56	165.42	7.07	0.92
		MOEA-SMPSO	60950.30	250.11	13.37	1.06
		MOEA-SMS-EMOA	64701.56	211.23	*__0.00__	1.04
		MOEA-E-NSGA-III	*__57568.73__	*__111.90__	*__0.00__	*__0.79__

* indicate the best value. The baseline is in italics.

Table 9: RPD-oriented metric values obtained for applications MPG and GMS. For all metrics, positive values represent favorable results (savings respecting *MOEA-NSGA-III*).

Application	Size	Autoscaler	Makespan RPD (%)	Cost RPD (%)	Task failures RPD (%)
MPG	30	MOEA-SMPSO	−58.10	8.19	−100.00
		MOEA-SMS-EMOA	1.65	−60.23	*100.00
		MOEA-E-NSGA-III	*2.46	*11.11	*100.00
	100	MOEA-SMPSO	−54.90	−135.59	−100.00
		MOEA-SMS-EMOA	−57.47	−175.45	*0.00
		MOEA-E-NSGA-III	*10.90	*14.86	*0.00
	300	MOEA-SMPSO	*17.77	*19.28	−195.00
		MOEA-SMS-EMOA	−25.24	−69.56	5.00
		MOEA-E-NSGA-III	16.56	18.18	*100.00
GMS	30	MOEA-SMPSO	−32.44	14.11	2.05
		MOEA-SMS-EMOA	*3.78	−28.07	−23.97
		MOEA-E-NSGA-III	2.19	*23.50	*36.76
	100	MOEA-SMPSO	−35.30	*6.88	−61.20
		MOEA-SMS-EMOA	−5.66	−32.07	51.80
		MOEA-E-NSGA-III	*10.37	5.57	*54.36
	300	MOEA-SMPSO	−32.50	6.97	−203.20
		MOEA-SMS-EMOA	−32.73	−40.27	−89.36
		MOEA-E-NSGA-III	*10.11	*15.52	*33.99

* indicate the best value.

Table 10: RPD-oriented metric values obtained for applications FPA and WRF. For all metrics, positive values represent favorable results (savings respecting *MOEA-NSGA-III*).

Application	Size	Autoscaler	Makespan RPD (%)	Cost RPD (%)	Task failures RPD (%)
FPA	30	MOEA-SMPSO	−40.68	−125.58	−100.00
		MOEA-SMS-EMOA	0.81	−379.07	*0.00
		MOEA-E-NSGA-III	*0.90	*6.98	*0.00
	100	MOEA-SMPSO	−67.08	−58.69	−100.00
		MOEA-SMS-EMOA	*2.43	−57.28	*0.00
		MOEA-E-NSGA-III	1.78	*15.96	*0.00
	300	MOEA-SMPSO	−29.28	−43.70	−100.00
		MOEA-SMS-EMOA	−33.65	−93.41	*0.00
		MOEA-E-NSGA-III	*4.09	*9.17	*0.00
WRF	30	MOEA-SMPSO	−1.07	11.92	−100.00
		MOEA-SMS-EMOA	−3.15	*40.07	*0.00
		MOEA-E-NSGA-III	*7.66	11.56	*0.00
	100	MOEA-SMPSO	−5.17	−17.06	−1042.50
		MOEA-SMS-EMOA	−5.57	*47.87	−732.50
		MOEA-E-NSGA-III	*3.14	39.32	*100.00
	300	MOEA-SMPSO	−1.69	−51.20	−89.11
		MOEA-SMS-EMOA	−7.95	−27.69	*100.00
		MOEA-E-NSGA-III	*3.95	*32.35	*100.00

* indicate the best value.

Likewise, the average cost RPD and the average number of task failures RPD metrics allow us to calculate the % difference between the average cost and the average number of task failures, respectively, of MOEA-E-NSGA-III (or MOEA-SMPSO, or MOEA-SMS-EMOA) in respect of the average cost (or the average number of tasks failures) of MOEA-NSGA-III.

From now on, for simplicity, we will refer to the average makespan, average monetary cost, and the average number of task failures just as makespan, monetary cost, and a number of task failures, respectively.

5.4.1. Results and Discussion. From the results presented in Tables 7–10, we can mention the following: regarding the makespan, MOEA-E-NSGA-III beat MOEA-NSGA-III in all cases, achieving better gains (~10%–17%) for the PSEs of the molecular dynamics area, i.e., MPG and GMS. Furthermore, it is important to note that for these PSEs (MPG and GMS), in the cases in which the lowest makespan was not reached, the obtained gain with respect to MOEA-NSGA-III is very close to the algorithms that obtained the highest percentage (with a difference of approximately 1.5%). Achieving a lower makespan in the context of these PSEs not only allows results

to be obtained in less time, but knowing these makespan reductions can allow disciplinary users to make decisions about whether it is convenient to invest those time gains in exploring more parameter values and thus obtaining greater precision in their results. Then, in the case of the PSEs of the meteorology area (FPA, WRF), the reductions in the makespan are very important since they allow the meteorologists to speed up the processing of the results. Obtaining the results of a prediction in less time allows farmers to make better decisions to activate defense methods and avoid damage if, for example, frost or some other type of harmful phenomenon occurs for people or farms.

Besides, MOEA-E-NSGA-III obtained better makespan values than MOEA-SMS-EMOA in 10 of the 12 cases (i.e., MPG-30, MPG-100, MPG-300, GMS-100, GMS-300, FPA-30, FPA-300, WRF-30, WRF-100, WRF-300), and better makespan values than MOEA-PSO in 11 of the 12 cases (i.e., MPG-30 and MPG-100; GMS-30, GMS-30, GMS-100, and GMS-300 tasks; FPA-30, FPA-100, and FPA-300; and WRF-30, WRF-30, WRF-100, and WRF-300).

In relation to the monetary cost, MOEA-E-NSGA-III outperformed MOEA-NSGA-III in all cases, providing good monetary cost savings (10%–25%) in 7 cases and very good monetary cost savings (30%–40%) in 2 cases. More specifically, the cost gains obtained by MOEA-E-NSGA-III for each of the applications are distributed as follows: in the case of the PSEs from the dynamic molecular area, the gains vary between approximately (11%–18%) for MPG and between (6%–23.5%) for GMS. Then, in the case of the meteorological applications, the gains obtained by MOEA-E-NSGA-III varied between (7%–16%) for FPA and between (11%–40%) for WRF. It is important to mention that for all these applications, cost reductions are very important since they allow users to make decisions regarding the possibility of acquiring more instances to execute the PSEs. Acquiring a greater number of instances to execute this type of application would imply greater parallelism, and, as a consequence, greater reductions in the makespan could be achieved if necessary.

From Tables 9 and 10 it can also be seen that MOEA-E-NSGA-III has obtained much better monetary cost values than MOEA-SMS-EMOA in 10 of the 12 cases (i.e, MPG-30, MPG-100 and MPG-300, GMS-30, GMS-100 and GMS-300, FPA-30, FPA-100 and FPA-300, and WRF-300), and much better monetary cost values than MOEA-PSO in 9 of the 12 cases (i.e., MPG-30 and MPG-100, GMS-30 and GMS-300, FPA-30, FPA-100 and FPA-300, and WRF-100 and WRF-300).

In relation to the number of task failures, MOEA-E-NSGA-III also reached a better performance than MOEA-NSGA-III in 7 cases (i.e., MPG-30 and MPG-300, GMS-30, GMS-100 and GMS-300, and WRF-100 and WRF-300), and the same performance as MOEA-NSGA-III in the remaining 5 cases. In the aforementioned 7 cases, MOEA-E-NSGA-III achieved very good savings regarding the number of task failures of MOEA-NSGA-III (33%–54% in 3 cases, and 100% in 4 cases). In addition, MOEA-E-NSGA-III significantly outperformed MOEA-SMPSO in the 12 cases and had a much better performance than MOEA-SMS-EMOA in 5 cases (i.e., MPG-300, GMS-30, GMS-100, and GMS-300, and WRF-100). Note that the fact that failures in the execution of tasks are reduced directly implies, as we described in the previous paragraphs, a potential impact on the makespan of the PSEs and their monetary cost.

Regarding the average L_2-norm metric, the value reached by MOEA-E-NSGA-III is better than that obtained by MOEA-NSGA-III in all cases. This is because MOEA-E-NSGA-III has obtained savings regarding the makespan and monetary cost of MOEA-NSGA-III in the 12 cases. Moreover, MOEA-E-NSGA-III has obtained savings regarding the number of task failures of MOEA-NSGA-III in 7 cases, and the same number of task failures of MOEA-NSGA-III in 5 cases.

In addition, the values obtained by MOEA-E-NSGA-III regarding the L_2-norm metric are much better than those of both MOEA-SMS-EMOA and MOEA-SMPSO, in all cases. This is because MOEA-E-NSGA-III performed better than MOEA-SMS-EMOA regarding the makespan and monetary cost in 10 cases and performed better than (equal to) MOEA-SMS-EMOA with respect to the number of task failures in 5 (7) cases. Besides, MOEA-E-NSGA-III performed better than MOEA-SMPSO respecting the makespan in 11 cases, the monetary cost in 9 cases, and the number of task failures in all 12 cases.

Finally, based on the performed analysis of the results, it can be concluded that the variant MOEA-E-NSGA-III can be considered the best alternative to reduce the makespan, cost, and failures inherent to the studied applications' execution. The use of the variant MOEA-E-NSGA-III would enable a positive impact on the execution of the described PSEs since it would allow its disciplinary users to obtain the results in less time and at a lower monetary cost, speeding up the analysis of results for decision-making (for example, deploying alert systems in the meteorology area).

5.4.2. Statistical Analysis. In order to determine whether the enhancement reached by MOEA-E-NSGA-III regarding MOEA-NSGA-III, MOEA-SMS-EMOA, and MOEA-SMPSO is significant, we carried out a significance test based on the obtained results in relation to the L_2-norm metric, specifically the Mann–Whitney U test [38]. As was previously mentioned, this metric trades off the three optimization objectives. For this reason, as mentioned in [14], this metric is suitable for carrying out the test. Regarding the results obtained by the evaluations of the variants in relation to the L_2-norm, it is necessary to recall that each variant was evaluated 30 times for each PSE application and size, leading to 30 values for the L_2-norm per case. Then, we carried out the mentioned test on the results got by MOEA-E-NSGA-III and MOEA-SMS-EMOA in relation to each PSE and size. Finally, we carried out such test on the results obtained by MOEA-E-NSGA-III and MOEA-SMPSO, in relation to each PSE and size. In all the cases, we used $\alpha = 0.001$ (confidence level). Based on the performed tests, MOEA-E-NSGA-III achieved significant improvements with respect to MOEA-NSGA-III, MOEA-SMS-EMOA, and MOEA-SMPSO, in all the PSEs and sizes. It is necessary to mention that we

considered the Mann–Whitney U test because the L_2-norm metric values, in any case, do not follow a normal distribution, which was instead confirmed by applying the test Shapiro–Wilk with $\alpha = 0.001$.

Besides, to determine if the improvements achieved by MOEA-NSGA-III regarding MOEA-SMS-EMOA and MOEA-SMPSO are significant, we carried out a statistical significance test similar to that previously described for determining the significance of the enhancement achieved by MOEA-E-NSGA-III. Specifically, we carried out the Mann–Whitney U test using the L_2-norm metric values obtained by MOEA-NSGA-III and MOEA-SMS-EMOA. Then, we carried out the mentioned test on the results got by MOEA-NSGA-III and MOEA-SMPSO for the L_2-norm metric, with respect to each PSE and size. In these cases, we also applied the test with $\alpha = 0.001$. According to the tests carried out, MOEA-NSGA-III reached significant improvements with respect to MOEA-SMS-EMOA in all the applications and sizes, and reached significant improvements with respect to MOEA-SMPSO, in 11 of the applications and sizes.

5.4.3. Computation Time Analysis. In this subsection, we show in Table 11 the average running time (in seconds) required by the variants MOEA-NSGA-III, MOEA-E-NSGA-III, MOEA-SMPSO, and MOEA-SMS-EMOA, for each PSE application and size. Moreover, Table 12 shows the average computation time (in seconds) of each multiobjective optimization algorithm of each variant, for each PSE application and size. The four variants were run on a PC equipped with an AMD Ryzen 5 with six 2022 MHz cores, 16 GB of RAM, and SDD, running Manjaro. Moreover, the four variants, including the algorithms used by them, were implemented in Java 1.8.

As shown in Table 11, the variant MOEA-NSGA-III required the lowest average computation time in all the PSEs and sizes. This is mainly because, as shown in Table 12, the algorithm NSGA-III used by this variant obtained the lowest average computation time, in all the PSEs and sizes.

The variant MOEA-E-NSGA-III required the average computation time closest to that of MOEA-NSGA-III in all the PSEs and sizes. The main reason for this is that, as shown in Table 12, the algorithm E-NSGA-III utilized by MOEA-E-NSGA-III obtained an average computation time very close to that of the algorithm NSGA-III in all the PSEs and sizes. In this sense, note that, as described in Section 4.2, the algorithm E-NSGA-III is an extension of the algorithm NSGA-III. These algorithms have the same general behavior and utilize the same crossover, mutation, and selection processes. However, these algorithms differ regarding the generation of the initial population. Unlike NSGA-III, E-NSGA-III includes a number of extreme solutions in the initial population, and these solutions are defined according to the problem at hand.

On the other hand, the variant MOEA-SMS-EMOA required an average computation time that exceeds significantly those of the variants MOEA-NSGA-III, MOEA-E-NSGA-III, and MOEA-SMPSO, in all the PSEs and sizes. This is because, as we presented in Table 12, the algorithm SMS-EMOA used by MOEA-SMS-EMOA obtained an average computation time that was considerably higher than that of the algorithms NSGA-III, E-NSGA-III, and SMPSO in each of the PSEs and sizes. In this respect, it is necessary to note that, as described in Section 4.3, the algorithm SMS-EMOA bases on the hypervolume metric to select solutions. In each iteration of this algorithm, the set of candidate solutions to be selected is defined, and then the contribution of each one of the candidate solutions to the hypervolume of the mentioned set is computed. These calculations require a significant amount of computation time and therefore affect the total computation time of SMS-EMOA.

Based on the results presented in Tables 7–12, the variant MOEA-E-NSGA-III achieved significant improvements with respect to the variants MOEA-NSGA-III, MOEA-SMS-EMOA, and MOEA-SMPSO in all the PSEs and sizes, requiring a computation time close to that of MOEA-NSGA-III (i.e., the variant with the lowest computation time) in all the PSEs and sizes. Besides, MOEA-NSGA-III significantly outperformed MOEA-SMS-EMOA in all the PSEs and sizes and reached significant improvements regarding MOEA-SMPSO in 11 of the PSEs and sizes.

5.4.4. Pareto Sets. As above mentioned, the variant MOEA-E-NSGA-III achieved better performance than the other variants MOEA-NSGA-III, MOEA-SMS-EMOA, and MOEA-SMPSO in all the applications and sizes. Considering that these four variants only differ regarding the multiobjective optimization algorithm used for obtaining the Pareto set of scaling plans, we analyzed the quality of the Pareto sets provided by each of the algorithms utilized in these variants. Recall that, as mentioned in Section 4, each variant applies one scaling plan extracted from the Pareto set of the algorithm being used. Thus, the quality of the Pareto sets provided by the algorithms impacts on the performance of these variants.

We focus on the sets generated by the algorithms during the first autoscaling stage. This is due to the fact that for each of the PSEs and sizes, this is the simulation window where all variants approach exactly the same multiobjective optimization problem, and the algorithms of these variants provide Pareto sets to solve this problem. Thereby, it is a fair comparison. During each of the subsequent autoscaling stages, the variants typically approach different optimization problems, which are defined considering the PSE' tasks execution state and also the virtual infrastructure state.

We used the well-known hypervolume metric [39], which is usually utilized in the literature to evaluate and compare Pareto sets. This metric calculates the volume of the objective space dominated by a given Pareto set and quantifies (a) how close this set is to the optimal Pareto set and (b) how well solutions in the set are distributed considering the objective space. It is necessary to mention that, in order to apply this metric, we considered that the volume of the objective space is bounded by the worst possible value of each of the considered optimization objectives.

Table 13 presents the average hypervolume of the Pareto sets provided by E-NSGA-III, NSGA-III, SMS-EMOA, and

Table 11: Average running time (seconds) of each variant of the autoscaler MOEA.

Application	Size	MOEA-NSGA-III	MOEA-E-NSGA-III	MOEA-SMPSO	MOEA-SMS-EMOA
MPG	30	**4**	6	7	670
	100	**20**	31	54	1767
	300	**110**	156	258	3270
GMS	30	**7**	10	13	890
	100	**37**	40	88	1571
	300	**210**	299	352	4271
FPA	30	**4**	**4**	5	430
	100	**24**	27	33	1221
	300	**80**	101	244	2683
WRF	30	**4**	6	11	889
	100	**30**	43	67	2277
	300	**129**	291	337	3613

The significance level of bold values given in Table 11 is $\alpha = 0.001$ (significance level).

Table 12: Average running time (seconds) by each algorithm of each variant of the autoscaler MOEA.

Application	Size	NSGA-III	E-NSGA-III	SMPSO	SMS-EMOA
MPG	30	**3**	4	7	585
	100	**13**	17	26	1530
	300	**41**	50	77	1872
GMS	30	**2**	4	5	616
	100	**14**	15	32	1065
	300	**74**	81	155	1784
FPA	30	**3**	**3**	4	359
	100	**15**	16	19	1109
	300	**43**	54	126	2278
WRF	30	**2**	3	5	626
	100	**15**	20	31	1355
	300	**56**	80	113	1421

Table 13: Average hypervolume of the Pareto sets provided by each algorithm (first autoscaling stage) for each of the applications and sizes. Higher values represent better values.

Application	Size	NSGA-III	SMPSO	SMS-EMOA	E-NSGA-III
MPG	30	67	26	43	**74**
	100	84	27	23	**94**
	300	33	40	12	**54**
GMS	30	54	51	38	**70**
	100	49	28	42	**62**
	300	70	48	44	**81**
FPA	30	94	48	56	**96**
	100	83	22	71	**87**
	300	76	38	29	**81**
WRF	30	17	14	18	**29**
	100	24	13	22	**34**
	300	18	14	16	**31**

The significance level of bold values given in Table 13 is $\alpha = 0.001$ (significance level).

SMPSO, during the first autoscaling stage, for each of the PSEs and sizes. As shown in Table 13, the algorithm E-NSGA-III has reached an average hypervolume value higher than those reached by the other three algorithms for each of the PSEs and sizes. This means that the Pareto sets provided by the algorithm E-NSGA-III are better than those of the other three algorithms in terms of both optimal Pareto set proximity and distribution of the scaling plans. Because of this, the variant MOEA-E-NSGA-III has been able to select and apply better scaling plans, and therefore outperform the other three variants, in all the PSEs and sizes.

6. Related Work

Studying the autoscaling problem [40] has received significant attention in the last ten years [12, 41–45]. However, these approaches differ in many aspects (see Table 14), including the type of application for which they were proposed, the optimization algorithms implemented, if both the scaling and scheduling problems were considered, the pricing model used (e.g., only on-demand, or on-demand and spot), the optimization objectives considered, and finally, the number of applications with which the approaches were tested. The next subsections are organized according to the type of application considered in each surveyed work (bag of tasks, web, or workflow application).

6.1. Bag-of-tasks (BoT) Applications. There are very few works that address the autoscaling problem and that are focused on bag-of-tasks type applications such as PSEs, and, at the same time, exploit spot instances to save costs. Among the surveyed works, we can mention a paper of our own [12], where an NSGA-II-based autoscaler called MOEA was proposed. The MOEA autoscales such kind of instances while reducing the makespan, monetary cost, and failures. However, the failures are not fully avoided, and therefore, the unexpected termination of some instances affects the completion time of their associated tasks, since these latter must be run in other instances. Then, in [14], MOEA was extended exploiting the use of NSGA-III for improving its performance with respect to the same considered optimization objectives. As a result, the NSGA-III-based autoscaler [14] has significantly outperformed the NSGA-II-based autoscaler [12], in terms of the makespan, cost, and number of task failures caused by the use of spot instances for two

Table 14: Relevant related works summary.

Application type	Paper	Algorithm	Autoscaling	Pricing model	Metrics	Number of applications
BoT	[12]	NSGA-II	Yes	On-demand, spot	Makespan, cost, and OOB errors	2
	[14]	NSGA-III	Yes	On-demand, spot	Makespan, cost, and OOB errors	2
	[46]	PSO - NN	No	On-demand, spot	Makespan and cost	1
	[47]	DEA	No	On-demand	Makespan and load balancing	1
	[42]	Check-pointing	No	On-demand, spot	Cost	1
	[43]	DDS	No	On-demand	Cost	3
	[48]	NSGA-III	Yes	On-demand	Makespan and cost	2
Web	[49]	Cost-efficient and fault tolerant	Yes	On-demand, spot	Availability, cost, and RT	1
	[50]	RLPAS	Yes	On-demand	CPU utilization, RT, and throughput	3
	[45]	RHAS	Yes	On-demand	Cost, RT, and QoS	2
	[51]	ML based on reactive and proactive	Yes	On-demand	Broker profit and cost	1
	[52]	NN—LR	Yes	On-demand	RT and cost	3
Workflows	[53]	SIAA	Yes	On-demand, spot	Makespan, cost, and task failures	4
	[54]	Online cost-efficient scheduling	No	On-demand, spot	Cost	4
	[55]	Dynamic approach	No	On-demand, spot	Cost, reliability, and fault tolerance	1
	[41]	Dynamic autoscaling based on EDF	Yes	On-demand	Cost	3
	[56]	Scaling first	Yes	On-demand	Cost	3
	[57]	Dynamic approach	Yes	On-demand	Makespan and cost	1
	[58]	NSGA-II	Yes	On-demand and spot	Makespan, cost, and OOB errors	4

different real PSE applications. A distinction of this article with respect to [12, 14] is that in this work we perform the autoscaling problem considering three new multiobjective optimization algorithms (i.e., E-NSGA-III, SMS-EMOA, and SMPSO), obtaining significant performance gains, in particular with our adapted E-NSGA-III.

Then, in [46], an efficient and cost-optimized scheduling algorithm for a BoT was proposed. The authors have used a particle swarm optimization (PSO) algorithm combined with an artificial neural network (NN) for load balancing and predicting the price of spot instances. The predicted prices are validated in relation to the current prices of spot instances, with the aim of minimizing both the time and monetary costs. Moreover, in [47], a population-based approach inspired by the differential evolution algorithm (DEA) was proposed to reduce makespan and improve load balancing. The approach is based not only on maintaining the diversity of the population but also on increasing the probability of searching for approximate optimal solutions. However, it is worth mentioning that although in [46, 47] the authors focused on BoT applications, they address task scheduling and not autoscaling, and besides, in [47] the authors have not considered the use of spot instances.

Then, in [42], the authors presented an approach for producing elastic clusters from computer resources coming from multiple CSPs. Concretely, the work deals with hybrid clouds (on-premise and public clouds). The proposed strategy [42] exploits spot instances to reliably deliver low execution costs. For this, a check-pointing algorithm was implemented for periodically keeping track of tasks' progress before the spot instance is terminated by the CSP. Thereby, the strategy supports resuming tasks from the last checkpoint. A study case based on the nonlinear dynamic analysis of buildings was performed to test the performance algorithm. On the other hand, in [43], a delay-based dynamic scheduling (DDS) for resource provisioning was proposed with the aim of minimizing the monetary cost and meeting the deadline constraint. For this, at runtime, new instances are allocated by the DDS component, considering the application state and estimated task execution times. While both approaches consider independent task applications and also have monetary cost as an optimization objective, contrary to this work, they are not based on metaheuristics.

Finally, in earlier work, we proposed a multiobjective intelligent autoscaler called MIA [48]. MIA is based on NSGA-III for executing PSEs in public clouds. Its goal is to

minimize makespan and monetary costs, but, unlike this work, it does not consider the use of spot instances.

6.2. Web Applications. In this subsection, we describe the approaches that focus on cloud-hosted web applications. Among the works that, like our proposal, consider the use of spot instances, we can mention the ones proposed in [49, 50]. In [49], the goal is to achieve high availability and minimize both the monetary cost and response time (RT), even when the CSP unexpectedly finalizes some spot instances. Specifically, a cost-efficient autoscaling and a fault-tolerant algorithm for further overprovisioning the same resource capacity by the use of another spot instance type were implemented. Accordingly, an application can tolerate some instances of finalization and remain fully provisioned. On the other hand, in [50], a reinforcement learning-based strategy called RLPAS was proposed to automatically scale the virtual infrastructure in a cloud. The goal consisted of minimizing response time and maximizing resource utilization and throughput. Concretely, RLPAS allows learning the cloud environment's resource state in parallel where there are heterogeneous and fluctuating workloads.

There are other works that only consider on-demand instances. In [45], a robust hybrid auto-scaler (RHAS) was proposed to reduce monetary costs and response time. The objective is to estimate the needed resources for horizontal scaling depending on the incoming workloads. Moreover, [51] presents a machine learning (ML)-based proactive algorithm combined with a reactive algorithm for scaling resources according to user's demands. The strategy, based on a price model, aims at both maximizing broker's profit and minimizing the user's costs. Besides, the combined strategy explores the scale-up condition that, in an autoscaling environment that is purely reactive, is used to rent more instances. Then, in the work presented in [52], an autoscaler called MLscale was proposed. The autoscaler does not require much knowledge of the application or manual tuning. MLscale uses NN to build the model of the application's performance in an online mode and multiple linear regression (LR) for predicting the state of the postscaling system. Besides, MLscale can accurately model the response time while minimizing the cost of resources for web applications. All the approaches presented in [45, 49–52] were focused on Web applications, where the requirements of individual tasks are much lighter than the PSEs considered in our article.

6.3. Workflows. Regarding works that consider workflows and applications and the use of spot instances, we can mention the work presented in [53], where a heuristic-based autoscaler was proposed for minimizing the makespan. The autoscaler, called SIAA, is subject to budget constraints, and the main distinction regarding our article is that in [53] the monetary cost was not taken into account and, moreover, was based on a heuristic. Another relevant work is [54], where a cost-efficient scheduling strategy for executing workflows was proposed. This strategy is subject to deadline constraints, and workflow tasks are scheduled on the spot instances. Then, in the work presented in [55], the authors studied how to efficiently run large-scale workflows on clouds, using on-demand and spot instances provided by the EC2 service of Amazon. In [55], both the spot price and the effect of spot instances disturbance were analyzed for designing a dynamic strategy able to minimize cost, increase reliability, and minimize the complexity of fault tolerance while maintaining overall performance and scalability.

Moreover, in [41], a dynamic strategy based on the earliest deadline first (EDF) subject to deadline constraints for efficiently executing multiple workflows was presented. The main objective consisted of ensuring that every single workflow task terminated before its deadline. Subsequently, the same authors extended the problem to consider budget constraints [56]. On the other hand, in the work presented in [57], an autoscaler that learns over time the tasks' resource needs from the workflow structure and automatically adapts the number of needed instances was proposed. The autoscaler must meet all task deadlines without having prior information about the workflow structure or execution times. The strategy was implemented to minimize the timeline and cost. Finally, in [58], the authors presented an online Cloud Multiobjective Intelligence (CMI) autoscaler for minimizing the duration, cost, and impact of the interruptions due to exploiting spot instances. This autoscaler is subject to budget constraints, and its goal is to periodically solve the cloud autoscaling problem while the workflow is executed.

Note that, from the surveyed works, there are few third-party approaches that deal with the autoscaling problem by using strategies based on multiobjective metaheuristics that consider the use of on-demand and spot instances while minimizing the makespan, monetary cost, and task failures of PSEs applications. In this paper, we also aim at addressing the significance of the results by evaluating the autoscaler via 4 (four) different applications and running statistical significance tests.

7. Conclusions and Future Research

The autoscaler MOEA is a recent cloud autoscaler based on the NSGA-II algorithm, which has been proposed to execute PSEs in public cloud environments. This autoscaler considers the well-known pricing models on-demand and spot in order to acquire VM instances for executing the tasks of a given PSE application. Besides, MOEA considers three optimization objectives relevant for the users: minimizing the computing time, the monetary cost, and the spot instances interruptions of the application's execution. However, the performance of this autoscaler with respect to the three considered optimization objectives depends significantly on the Pareto set of scaling plans provided by the multiobjective optimization algorithm used. As detailed in [14], the performance of this autoscaler improves considerably when the algorithm NSGA-II is replaced by NSGA-III. However, the algorithm NSGA-III has limitations in terms of the diversity of the resulting Pareto set, which can negatively impact on the performance of the autoscaler.

Motivated by this, we have analyzed the incorporation of other multiobjective optimization algorithms into the autoscaler MOEA in order to enhance the performance of

this autoscaler with respect to the three considered optimization objectives. In this sense, we incorporated the following three popular of such kind of algorithms: E-NSGA-III, SMS-EMOA, and SMPSO. These algorithms have important behavioral differences w.r.t. NSGA-III, as utilized in [14]. The obtained autoscaler variants were referred to as MOEA-E-NSGA-III, MOEA-SMS-EMOA, and MOEA-SMPSO. Besides, the variant of the autoscaler with the algorithm NSGA-III [14] was considered as a reference for comparison purposes and referred to as MOEA-NSGA-III.

We evaluated each of the four variants of the autoscaler MOEA on four real-world PSE applications from the molecular dynamics and meteorology areas. We considered three sizes per application, i.e., the number of tasks involved in an application. Besides, we considered the characteristics of on-demand and spot instances which correspond to five VM types available in Amazon EC2. We considered the aforementioned applications and characteristics of VM instances with the aim of defining diverse realistic experimental scenarios. The evaluations of the four variants on the applications were developed by using the well-known CloudSim simulator [22].

After that, we compared the performance of the four variants of the autoscaler MOEA on all the PSEs and sizes, regarding the three optimization objectives considered. According to the performance comparison developed, the variant MOEA-E-NSGA-III outperformed the other three variants with regard to the L_2-norm metric in all the PSEs and sizes. In particular, the variant MOEA-E-NSGA-III achieved better values than the variant MOEA-NSGA-III in relation to the average makespan, monetary cost, and number of task failures inherent to spot instance interruptions in all the PSEs and sizes. Since the four variants only differ regarding the multiobjective optimization algorithm used to obtain the Pareto set of scaling plans, we analyzed the quality of the Pareto sets provided by each of the algorithms utilized in these variants via the hypervolume metric. The obtained hypervolume values indicated that the Pareto sets provided by the algorithm E-NSGA-III are better than those provided by the other three algorithms, in respect of optimal Pareto set proximity and solution distribution. Because of this, the variant MOEA-E-NSGA-III has been able to apply better scaling plans, and thus outperform the other three variants in all the PSEs and sizes. Additionally, we have compared the performance of the four variants on each PSE and size in relation to the required average computation time. Regarding this, the variant MOEA-E-NSGA-III required an average computation time very close to that of MOEA-NSGA-III (i.e., the variant with the lowest average computation time) in all the PSEs and sizes. The computation time needed by MOEA-E-NSGA-III represents a small percentage (i.e., less than 4.5% in most of the applications and sizes used) of the computation time that corresponds to one autoscaling stage (i.e., one hour).

The results obtained by the variant MOEA-E-NSGA-III are encouraging. The reason is that reducing the three considered optimization objectives would have a positive impact on the execution of PSEs, for example, by speeding up the analysis of results for decision-making (e.g., issuing a weather alert). In addition, users could take advantage of the reductions in monetary costs to invest in acquiring more instances, explore a greater number of parameters, and thus obtain greater precision in the results.

Considering the aforementioned facts, we conclude that the variant MOEA-E-NSGA-III represents a better autoscaling alternative compared to MOEA-NSGA-III for solving diverse instances of the addressed multiobjective cloud autoscaling problem.

Regarding the addressed multiobjective cloud autoscaling problem, it is necessary to note that this problem aims to decide the scaling plan to be applied in each autoscaling stage. Then, the scheduling of the PSE's tasks on the VM instances indicated in the scaling plans is solved by applying a well-known scheduling algorithm named ECT. However, considering that ECT is a greedy algorithm, the schedule provided by this algorithm in each autoscaling stage could not significantly approximate the optimal schedule. Therefore, a future research line involves modeling a new variant of the problem to simultaneously decide on the scaling plan and the scheduling plan so that the optimization objectives are achieved. In this new variant of the addressed problem, we will also incorporate other optimization objectives relevant to the context of cloud environments. In particular, we will consider energy consumption [59]. Since PSEs require a large number of computational resources to run efficiently, energy consumption has become a crucial problem [60] due to the high costs of electricity and CO_2 emissions. Therefore, it is also important to minimize energy consumption.

In the future, we will also incorporate other optimization algorithms into the first autoscaling phase of MOEA. In the first place, we will analyze the incorporation of other multiobjective optimization algorithms with the goal of improving the performance of the variant MOEA-E-NSGA-III. In particular, we will consider adaptive multiobjective evolutionary algorithms [39], which adapt their behavior (i.e., crossover, mutation, and selection processes) according to the evolutionary search state in order to promote the exploration/exploitation of the search space and thus improve the quality of the resulting Pareto set. Note that, unlike this kind of algorithm, the algorithm E-NSGA-III does not adapt its behavior according to the evolutionary search state. Therefore, we consider that this kind of algorithm could obtain better Pareto sets than those obtained by E-NSGA-III. As a result, the incorporation of this kind of algorithm into the autoscaler MOEA could outperform the performance of MOEA-E-NSGA-III. In the second place, we will design and incorporate multiobjective optimization algorithms aimed at addressing the new variant of the cloud autoscaling problem.

In this line, PSE applications considered in this paper are not real-time applications (i.e., they do not require results before a predefined deadline). Thus, in the experiments developed, restrictions about the optimization delay were not considered. However, with respect to the unavoidable computing uncertainty as conceptualized in [61], there are some improvements to be made. We are working on an approach to sample crucial points in the input parameter

space used to execute a given PSE, so we can streamline the execution of the tasks associated with such points and the visualization of results. In this way, users could partially visualize PSE output variables earlier instead of waiting for all PSE tasks to be completed.

Finally, notice that the VM instances provided by public CSPs have other sources of uncertainty. For example, there may be variability in performance due to the use of virtualization technologies, the concurrent use of multiple users, and possible failures. According to studies in [62], the performance variability of VM instances is about 20%. Therefore, to deal with this uncertainty, future research will consist of addressing the cloud autoscaling problem through deep learning (DL) techniques. DL techniques will allow us to perform predictions on infrastructure performance and thus make better decisions regarding how it is convenient to scale the infrastructure and improve the global performance to execute PSEs.

Data Availability

The datasets generated and/or analyzed in the study are available upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

Conceptualization was done by V. Y., E. P., D. M., and C. M.; data curation was provided by V. Y. and E. P.; formal analysis was performed by V. Y.; the investigation was done by V. Y., E. P., E. M., J. S., and C. M.; methodology was handled by V. Y., E. P., and C. M.; project administration was provided by V. Y., E. P., C. M., and G. R.; software analysis was done by V. Y., D. M., and C. M.; supervision was provided by V. Y., C. M., and G. R.; validation was done by V. Y. and C. M.; visualization was performed by V. Y., E. P., and C. M.; and writing the original draft was performed by V. Y., E. P., and C. M. All authors have checked the manuscript and have agreed to the submission and the specified author order.

References

[1] 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.

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

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

[4] S. Suram, N. A. MacCarty, and K. M. Bryden, "Engineering design analysis utilizing a cloud platform," *Advances in Engineering Software*, vol. 115, pp. 374–385, 2018.

[5] J. Rogeiro, M. Rodrigues, A. Azevedo et al., "Running high resolution coastal models in forecast systems: moving from workstations and HPC cluster to cloud resources," *Advances in Engineering Software*, vol. 117, pp. 70–79, 2018.

[6] D. Kumar, G. Baranwal, Z. Raza, and D. P. Vidyarthi, "A survey on spot pricing in cloud computing," *Journal of Network and Systems Management*, vol. 26, no. 4, pp. 809–856, 2018.

[7] C. W. Tsai and J. J. P. C. Rodrigues, "Metaheuristic scheduling for cloud: a survey," *IEEE Systems Journal*, vol. 8, no. 1, pp. 279–291, 2014.

[8] T. Lorido-Botran, J. Miguel-Alonso, and J. ., A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.

[9] A. Barnawi, S. Sakr, W. Xiao, and A. Al-Barakati, "The views, measurements and challenges of elasticity in the cloud: a review," *Computer Communications*, vol. 154, pp. 111–117, 2020.

[10] S. Verma and A. Bala, "Auto-scaling techniques for IoT-based cloud applications: a review," *Cluster Computing*, vol. 24, no. 3, pp. 2425–2459, 2021.

[11] Y. Garí, D. A. Monge, E. Pacini, C. Mateos, and C. García Garino, "Reinforcement learning-based application Autoscaling in the Cloud: a survey," *Engineering Applications of Artificial Intelligence*, vol. 102, pp. 104288–104323, 2021.

[12] D. A. Monge, E. Pacini, C. Mateos, and C. Garcıa 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.

[13] 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.

[14] V. Yannibelli, E. Pacini, D. Monge, C. Mateos, and G. Rodriguez, "A comparative analysis of NSGA-II and NSGA-III for autoscaling parameter sweep experiments in the cloud," *Scientific Programming*, vol. 2020, Article ID 4653204, 17 pages, 2020.

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

[16] H. Ishibuchi, R. Imada, Y. Setoguchi, and Y. Nojima, "Performance comparison of nsga-ii and nsga-iii on various many-objective test problems," in *Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC)*, pp. 3045–3052, Vancouver, BC, Canada, July, 2016.

[17] 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.

[18] 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 (ICMLA 2018)*, pp. 17–20, Orlando, FL, USA, December, 2018.

[19] P. Wangsom and K. Lavangnananda, "Extreme solutions NSGA-III (E-NSGA-III) for multiobjective constrained problems," in *Proceedings of the 2019 International Conference on Optimization and Learning (OLA 2019)*, Bangkok, Thailand, January, 2019.

[20] N. Beume, B. Naujoks, and M. Emmerich, "SMS-EMOA: multiobjective selection based on dominated hypervolume,"

European Journal of Operational Research, vol. 181, no. 3, pp. 1653–1669, 2007.

[21] A. J. Nebro, J. J. Durillo, J. Garcia-Nieto, C. A. Coello Coello, F. Luna, and E. Alba, "SMPSO: a new PSO-based metaheuristic for multiobjective optimization," in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM)*, pp. 66–73, Nashville, TN, USA, March, 2009.

[22] M. C. Silva Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inácio, and M. M. Freire, "CloudSim Plus: a cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness," in *Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 400–406, Washington, DC, USA, August, 2017.

[23] J. M. Framinan and P. Perez-Gonzalez, "New approximate algorithms for the customer order scheduling problem with total completion time objective," *Computers & Operations Research*, vol. 78, pp. 181–192, 2017.

[24] S. Akyol and B. Alatas, "Plant intelligence based metaheuristic optimization algorithms," *Artificial Intelligence Review*, vol. 47, no. 4, pp. 417–462, 2017.

[25] B. Alatas and H. Bingol, "Comparative assessment of light-based intelligent search and optimization algorithms," *Light & Engineering*, vol. 28, pp. 51–59, 2020.

[26] 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.

[27] D. S. Bertoldi, E. N. Millán, and A. J. Fernández Guillermet, "Phenomenology of the heating, melting and diffusion processes in Au nanoparticles," *Physical Chemistry Chemical Physics*, vol. 23, no. 2, pp. 1298–1307, 2021.

[28] S. K. Ghosh and T. Pal, "Interparticle coupling effect on the surface plasmon resonance of gold nanoparticles: from theory to applications," *Chemical Reviews*, vol. 107, no. 11, pp. 4797–4862, 2007.

[29] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, vol. 117, no. 1, pp. 1–19, 1995.

[30] S. Ali, V. S. Myasnichenko, and E. C. Neyts, "Size-dependent strain and surface energies of gold nanoclusters," *Physical Chemistry Chemical Physics*, vol. 18, no. 2, pp. 792–800, 2016.

[31] M. B. Planes, E. N. Millán, H. M. Urbassek, and E. M. Bringa, "Collisions between micro-sized aggregates: role of porosity, mass ratio, and impact velocity," *Monthly Notices of the Royal Astronomical Society*, vol. 503, no. 2, pp. 1717–1733, 2021.

[32] D. Rim, L. G. Moyano, and E. N. Millán, "Unsupervised machine learning algorithms as support tools in molecular dynamics simulations," *Simposio Argentino de Inteligencia Artificial*, vol. 48, 2019.

[33] F. Breque and M. Nemer, "Frosting modeling on a cold flat plate: comparison of the different assumptions and impacts on frost growth predictions," *International Journal of Refrigeration*, vol. 69, pp. 340–360, 2016.

[34] L. E. Iacono, J. L. Vázquez-Poletti, C. García Garino, and I. M. Llorente, "Performance models for frost prediction in public cloud infrastructures," *Computing and Informatics*, vol. 37, no. 4, pp. 815–837, 2018.

[35] R. L. Snyder and J. P. de Melo-Abreu, *Frost Protection: Fundamentals, Practice and Economics, Volume 1 of Environment and Natural Resources Series*, Food and Agriculture Organization of the United Nations (FAO), Rome, Italy, 2005.

[36] J. Jeffers, J. Reinders, and A. Sodani, "Chapter 22: weather research and forecasting (WRF)," in *Intel Xeon Phi Processor High Performance Programming*, J. Jeffers, J. Reinders, and A. Sodani, Eds., pp. 499–510, Morgan Kaufmann, Burlington, MA, USA, Second Edition, 2016.

[37] W. C. Skamarock, J. B. Klemp, J. Dudhia et al., *A Description of the Advanced Research WRF Version 4.3*, University Corporation for Atmospheric Research, Boulder, Colorado, 2021.

[38] 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.

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

[40] 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.

[41] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–12, Washington, DC, USA, November, 2011.

[42] 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.

[43] 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.

[44] Z. Lu, X. Wang, J. Wu, and P. C. Hung, "InSTechAH: cost-effectively autoscaling smart computing hadoop cluster in private cloud," *Journal of Systems Architecture*, vol. 80, pp. 1–16, 2017.

[45] P. Singh, A. Kaur, P. Gupta, S. S. Gill, and K. Jyoti, "RHAS: robust hybrid auto-scaling for web applications in cloud computing," *Cluster Computing*, vol. 24, no. 2, pp. 717–737, 2021.

[46] S. G. Domanal and G. R. M. Reddy, "An efficient cost optimized scheduling for spot instances in heterogeneous cloud environment," *Future Generation Computer Systems*, vol. 84, pp. 11–21, 2018.

[47] J. Chen, P. Han, Y. Liu, and X. Du, "Scheduling independent tasks in cloud environment based on modified differential evolution," *Concurrency and Computation: Practice and Experience*, vol. 33, Article ID e6256, 2021.

[48] V. Yannibelli, E. Pacini, D. Monge, C. Mateos, and G. Rodriguez, "An NSGA-III-based multiobjective intelligent autoscaler for executing engineering applications in cloud infrastructures," in *Advances in Soft Computing. MICAI 2020*, L. Martínez-Villaseñor, Ed., vol. 12468, pp. 249–263, Springer, Berlin, Germany, 2020.

[49] 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.

[50] 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. 1348–1363, 2018.

[51] A. Biswas, S. Majumdar, B. Nandy, and A. El-Haraki, "A hybrid auto-scaling technique for clouds processing applications with service level agreements," *Journal of Cloud Computing*, vol. 6, no. 1, 2017.

[52] M. Wajahat, A. Karve, A. Kochut, and A. Gandhi, "MLscale: a machine learning based application-agnostic autoscaler," *Sustainable Computing: Informatics and Systems*, vol. 22, pp. 287–299, 2019.

[53] 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.

[54] 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.

[55] 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 2013 International Conference on Parallel and Distributed Systems*, pp. 657–662, Seoul, South Korea, December, 2013.

[56] 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, Cambridge, MA, USA, May, 2013.

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

[58] D. A. Monge, E. Pacini, C. Mateos, E. Alba, and C. García Garino, "CMI: an online multiobjective genetic autoscaler for scientific and engineering workflows in cloud infrastructures with unreliable virtual machines," *Journal of Network and Computer Applications*, vol. 149, Article ID 102464, 2020.

[59] J. Chen, Y. He, Y. Zhang, P. Han, and C. Du, "Energy-aware scheduling for dependent tasks in heterogeneous multiprocessor systems," *Journal of Systems Architecture*, vol. 129, Article ID 102598, 2022.

[60] F. N. Al-Wesabi, M. Obayya, M. A. Hamza, J. S. Alzahrani, D. Gupta, and S. Kumar, "Energy aware resource optimization using unified metaheuristic optimization algorithm allocation for cloud computing environment," *Sustainable Computing: Informatics and Systems*, vol. 35, Article ID 100686, 2022.

[61] H. Chen, X. Zhu, G. Liu, and W. Pedrycz, "Uncertainty-aware online scheduling for real-time workflows in cloud service environment," *IEEE Transactions on Services Computing*, vol. 14, no. 4, pp. 1167–1178, 2021.

[62] J. Ericson, M. Mohammadian, and F. Santana, "Analysis of performance variability in public cloud computing," in *Proceedings of the 2017 IEEE International Conference on Information Reuse and Integration*, pp. 308–314, IEEE, San Diego, CA, USA, August, 2017.