Bidding Strategy Based on Adaptive Differential Evolution Algorithm for Dynamic Pricing IaaS Instances

Dawei Kong[1], Guangze Liu[2], Li Pan[1], and Shijun Liu[1](✉)

[1] School of Software, Shandong University, Jinan, China
wei2white@163.com, {panli,lsj}@sdu.edu.cn
[2] Pennsylvania State University, State College, PA, USA
gbl5120@psu.edu

Abstract. In recent years, with the development of cloud computing technology and the improvement of infrastructure performance, cloud computing has developed rapidly. In order to meet the diverse needs of users and to maximize the revenue of cloud computing service providers, cloud providers have launched auction-type instances like Amazon Spot instances in the AWS cloud. For dynamic pricing cloud instances, how to select appropriate instance or instance group among multiple instances and make reasonable bids to optimize its own costs is a great challenge. This paper models the dynamic pricing instance pricing and multi-instance combination problem as a constrained optimization problem. Then we introduce the basic differential algorithm and proposes an adaptive differential evolution algorithm to optimize the combination of price bidding based on the optimal cost and the use of instances. Finally, we use real dynamic pricing instance price data released by the Amazon cloud to verify the optimization strategy. The experimental results show that the adaptive differential evolution algorithm has a better optimization effect on short-term task requirements and long-term task requirements.

Keywords: Dynamic pricing model · Iterative algorithm · Amazon cloud · Spot instance

1 Introduction

This computing resource sharing pool is called "cloud". Cloud computing brings together many computing resources. In other words, the power of computing can be regarded as a commodity. Cloud service types can be divided into three categories, infrastructure as a service (IaaS), and platform as a service (PaaS) and software as a service (SaaS). IaaS is one of the main service categories. It provides virtual computing resources, such as virtual machines, storage, networks, and operating systems, to individuals or organizations of cloud computing providers. Encapsulating basic resources such as hardware devices into services for users to use, customers cannot manage or control the cloud infrastructure,

H. Gao et al. (Eds.): CollaborateCom 2020, LNICST 349, pp. 458–478, 2021.
https://doi.org/10.1007/978-3-030-67537-0_28

but can control the operating system, storage, and applications deployed by themselves, and can also partially control the network components used. For example, Amazon's EC2 [1] and Google Compute Engine [2].

In recent years, with the development of cloud computing technology and the improvement of infrastructure performance, cloud computing has developed rapidly. Not only in the professional research field related to cloud computing but also more and more companies and individuals are beginning to deploy their businesses and applications to the cloud. Users can quickly build large and complex network applications, which greatly reduces equipment costs by using cloud computing platforms. Cloud platform providers virtualize computing resources through virtualization technology. The user uses the instances provided by the cloud platform to pay for the use of resources in the cloud platform. The payment methods of cloud computing services mainly include on-demand payment and scheduled payment. On-demand instance type is which users pay corresponding fees based on the instance type and usage time. Scheduled instances are which users pay some fees in advance and then receive a discount on the price when using the instance. In addition, in order to meet the user's diversity of instance requirements and to maximize the benefits of cloud computing service providers, cloud providers have launched auction-type instances. Amazon Cloud has launched Spot instances in the AWS cloud. Compared with the price of on-demand instances, Spot instances receive an ultra-low discount on usage fees [3]. Unlike the fixed pricing of on-demand instances, Spot instances use dynamic pricing, and the dynamic price of Spot instances is called the hourly price of spot instances. The spot price of each instance type in each available region is set by Amazon EC2 and gradually adjusted according to the long-term supply and demand of Spot instances. As long as there is the available capacity and the maximum price per hour requested by the user exceeds the Spot price, each Spot instance will run [4]. Amazon Web Service (AWS) prices Spot instances based on the application of Spot instances and the resource situation of EC2 instances. However, Amazon has not announced the detailed pricing strategy of spot instances. When users use bidding instances, they cannot accurately predict the price changes of the instances in the future [5].

In addition to cloud computing providers providing cloud computing resources, how to define service charges is also a key issue. The rationality of the pricing method can greatly influence the choice of users. Both Amazon's EC2 and Google's GCE have defined detailed charging methods. In order to meet the diverse needs of users, in order to improve the utilization of system resources and maximize the revenue of cloud platforms, cloud providers are also constantly enriching the billing methods of cloud platforms. Spot instances with dynamic pricing.

When users apply for the use of Spot instances, they need to bid on the instances, and users need to bid based on their experience. This undoubtedly increases the difficulty of users using Spot instances. In order to let users referring to the auction price, Amazon released historical data on the price of each type of spot instance in the first 90 days. Therefore, it is possible to predict future price changes by investigating 90-day price data, which helps to formulate reasonable bidding strategies. Amazon adjusted its pricing strategy for Spot on November 17, 2018. Prior to this, the prices of all types of spot instances fluctuated greatly. Now Amazon Cloud has shifted from a short-term resource

application and pricing strategy for remaining resources to a pricing strategy based on long-term benefit. Therefore, the current prices of many spot instances have stabilized. After the price of Spot instances has stabilized, it is more convenient for users to make bids and formulate reasonable use strategies. It is meaningless to predict future prices for stable and constant price data. Through the research on the pricing historical data of Spot instances, it is found that in certain available areas, for certain types of instances, the price of Spot instances still has great volatility. Therefore, when users apply for instance resources, they often need to apply for multiple instances at the same time to meet the needs of the task. In this dynamic pricing instance model, how to combine resources to achieve the maximum profit bidding strategy is a very important problem. Resource optimization and scheduling problem belong to NP problems and NP problems are difficult to solve or time cost. Therefore, in our problem, we seek the approximate optimal solution of the problem using heuristic differential evolutionary algorithms. In this paper, we optimize mutation methods and adaptive parameters of the basic differential evolution algorithms. Experiment results show that our adaptive algorithm has a better result.

2 Related Work

Cloud computing has developed into a key information technology and system model. Cloud computing-related technologies have made great progress, and the scale of cloud computing has grown significantly, becoming a popular direction in the research field [6]. For research in the field of cloud computing, cloud pricing models and resource optimization have always been a hot research topic [7].

The Spot instance launched by Amazon cloud has very obvious characteristics in its bid payment method and dynamic price. Therefore, many scholars and research institutes have published a lot of research on Amazon's Spot instance pricing strategy, price prediction, and maximizing returns [8]. The physical resources of cloud computing providers often have a lot of idle resources, which causes a lot of waste of resources. In order to minimize the economic losses caused by idle resources, cloud providers need to incentivize users to use these idle resources [9]. Kaminski B and Szufel P analyzed the factors that control the price of Amazon EC2 spot instances and proposed an adaptive bidding strategy that can digest the cost of use. It also proves that the bidding close to the spot price and the dynamic switching between instances is an effective and easy to implement strategy [10].

Efficiently providing resources is a challenging problem in the cloud computing environment because it is dynamic and needs to support heterogeneous applications. Although VM (Virtual Machine) technology allows multiple workloads to run concurrently and use shared infrastructure, it still cannot guarantee application performance. Therefore, current cloud data center providers either do not provide any performance guarantees or prefer static VM allocation rather than dynamic allocation, which leads to inefficient utilization of resources. In addition, due to the execution of different types of applications (such as HPC and web), workloads may have different QOS (Quality of Service) requirements, which makes resource provision more difficult. Early work either focused on a single type of SLA (service level agreement) or on the resource usage pattern of applications (such as web applications), resulting in inefficient use of data center

resources. In this article, we deal with the problem of resource allocation in a data center that runs different types of application workloads, especially non-interactive and cross-operating system applications. Vivek H. Bharad and Hitesh A. Bheda proposed admission control and scheduling mechanism that maximizes resource utilization and profits while ensuring that user QOS requirements specified in the SLA are not met. In order to better provide and utilize data center resources, it is important to understand the different types of SLAs and the applicable penalty and workload combinations. Compared with static server integration, the proposed mechanism provides great improvements and reduces SLA violations [11].

The reference factor for user bidding is often learned from historical price data or bidding strategies. Some studies use the historical price data of bidding instances to record the change of the price of each instance from p_i to p_j as the state transition, as well as obtain the probability of the state transition from p_i to p_j statistically according to the historical data. Since each instance corresponds to the values of Ni prices, a Ni * Ni price probability transfer matrix [8] is obtained, which the problem is modeled as a Markov chain model and to be solved [12].

Many scholars have studied the optimization of parallel tasks [13, 14]. Differential Evolution Algorithm (DE) was first proposed by Storn and Price in 1995. It is mainly used to solve real number optimization problems [15]. This algorithm is a type of group-based adaptive global optimization algorithm, which is a type of evolutionary algorithm. Because of its simple structure, easy implementation, fast convergence, and strong robustness, it is widely used. In recent years, differential evolution algorithms have been used in constrained optimization calculation [16], clustering optimization calculation [17], and nonlinear optimization control [18]. Like the genetic algorithm [19], the differential evolution algorithm is also an optimization algorithm based on modern intelligence theory, which refers to the optimization of constrained optimization calculations and the direction of optimization search through the group intelligence generated by cooperation and competition among individuals within the group.

3 Dynamic Pricing IaaS Instances

In this section, we introduce dynamic pricing instances and the method of applying dynamic pricing IaaS cloud instances. Cloud platform users usually purchase IaaS cloud services in the form of VM, according to the payment of related fees, to run the user's tasks. Each cloud platform service provider platform (such as Amazon EC2) has varied types of VM (virtual machine) with multiple physical resource configurations. Such as c4.xlarge, c4.2large, and c4.4xlarge. Generally speaking, high-performance VM instances usually have a higher price than others. When users buy IaaS instances from the cloud platform, they can choose instance type (such as c4.xlarge), operating system (such as Unix or Windows), and other optional items according to the actual situation. In the model of this paper, users need to apply for instance resources based on IaaS, and each user needs to apply for a set of instances in order to complete the work in time. In fixed-price cloud instances, users only need to consider the pricing of the current instance and their budget to select the instance. For dynamic pricing instances, users need to give pricing and make strategic choices. As an example, in the Amazon Spot

instance scenario, the price of a Spot instance is affected by many factors. In addition to the performance of the instance, it is also affected by the number of users and the application status of the instance. Many times, we can see that better-performing instances have a lower price than lower-performing instances. In this case, the user needs to adjust the resource usage strategy to maximize the user's revenue.

3.1 Fixed Pricing Cloud Instance Model

The price of each fixed pricing cloud instances model is fixed and it does not change over time. So, users only need to consider the budget and the performance requirements of the instances. As shown in Fig. 1.

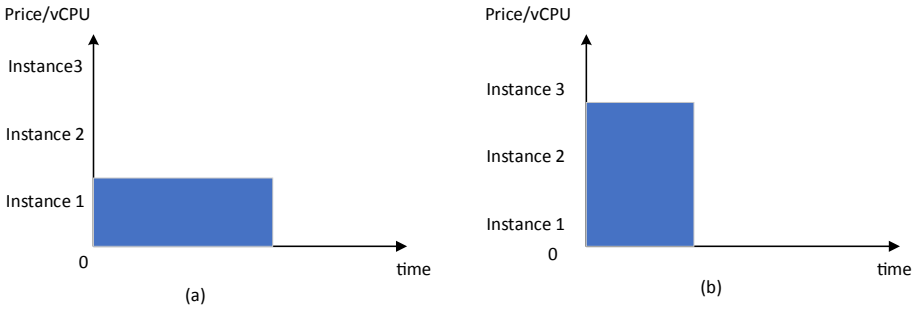

Fig. 1. Fixed pricing instances

In Fig. 1, Instance 1, Instance 2, and Instance 3 are three different performance instance types. C_i (capacity) is used to represent the performance of different instances, and Pi represents the price of different instances. Now there are $C_{Instance\ 3} > C_{Instance\ 2} > C_{Instance\ 1}$, $P_{Instance\ 3} > P_{Instance\ 2} > P_{Instance\ 1}$. The X-axis of the coordinate axis represents the time the task runs after the user selects an instance, and the Y-axis represents the price of different performance instances. The rectangular part of the figure represents the cost that the user needs to pay after selecting an instance. It can be seen that when the user submits the same task, regardless of the task due date, select a different instance type, and the actual fee paid Also different. In the fixed-price model, the pricing of each instance is fixed. The user selects an instance according to the budget and pays the actual fee according to the used price. This model is widely adopted by cloud providers due to its ease of use. For example, Amazon's on-demand instances use fixed pricing instance models. The advantage of this model is that the price of each instance is fixed, and the performance is stable. The user selects and pays a predictable fee according to the actual situation so that the budget can be well controlled while completing the task.

3.2 Cloud Instance Model for Dynamic Prices

The instance model based on bidding requires users to adopt competitive bidding methods to bid, and users need to specify the highest acceptable bid for a certain instance of

the application. Under normal circumstances, when the user's bid is higher than the cloud service provider's bid, the application instance gets run. If the user's bid is lower than the cloud service provider's pricing (the pricing is immediate and constantly changing), the bidding instance fails as shown in Fig. 2.

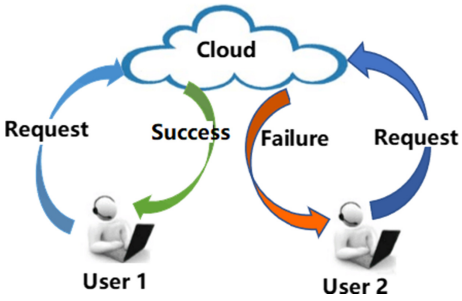

Fig. 2. Applying the bidding instance

The price of the bidding instance is specified by the cloud service provider. The cloud service provider will adjust the instance price according to the number of applications for the bidding instance, idle instance resource, and some other conditions. In this auction model, the price of cloud instances is usually dynamic. Most cloud service providers have launched bidding instances, such as Google Cloud's bidding instance, Alibaba Cloud's bidding instance, and Amazon's Spot instance.

3.3 Features of Amazon Spot Instances

Amazon's bidding instance is named Spot instance. Spot instance is Amazon's idle computing resources of Amazon EC2. Since the Spot instance is based on the bidding model, the price of the Spot instance fluctuates dynamically. Figure 3 shows the price data with the instance type is r3.8xlarge, the region is us-east-1, and the operating system is Linux.

As shown in Fig. 3, Amazon Cloud set the price of Spot instances in real-time. Since Amazon Cloud has not announced a detailed price strategy, it can only speculate that the price of Spot instances is related to the remaining capacity of Spot instances and the demand for Spot instances. In short, Amazon updates the price of Spot instances every 5 min. To use a Spot instance, the user needs to submit a Spot instance request, specify the type of instance, the region, and the highest price that the Spot instance can accept per hour. Users can propose bid prices based on historical price analysis of instance provided by Amazon. We use Amazon EC2 API and the AWS management console to obtain the historical price data of each Spot instance. When the user's highest bid price exceeds the current Spot instance's official pricing, the user's Spot instance application is accepted, and Spot instance is allocated to the user. The instance will continue to run until the user chooses to release it or Spot instance's official pricing is higher than the user's bidding price. According to the resource allocation strategy or revenue, Amazon Cloud will also have a certain probability of recovering the Spot instance applied by

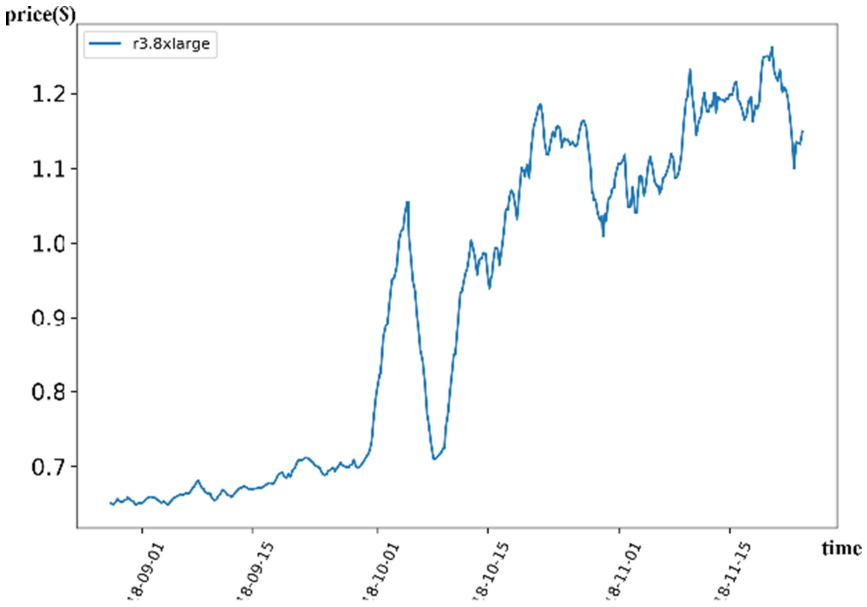

Fig. 3. A dynamic pricing instance

the user. In the dynamic pricing model of Spot instances, users cannot simply make the optimal resource application plan as easily as using fixed pricing instances. Users need to give a reasonable bidding strategy to adjust their workload. It is difficult to find the best answer to optimizing strategies.

4 Dynamic Bidding Modeling

4.1 Dynamic Bidding Modeling

Users apply for a dynamic price instance and give a bid price before using the dynamic pricing instances. If users bidding price is higher than the cloud provider's pricing, the instances are successfully allocated and executed. While these instances are running for user tasks, there are still many situations that they will be terminated by the cloud management system. Such as Amazon Spot instance, when the Spot instance capacity is insufficient, or the price rises above the bid, the applied instance will be terminated by the cloud management system. Amazon Cloud has not announced a specific price strategy, therefore, for users, there may be unexpected instances that may be terminated. The possibility of this kind of termination case that approximates the random nature greatly increases the difficulty of the relevant research on the auction case. Therefore, the research problem in this paper does not consider the situation where the cloud provider has been re-managed due to internal policy adjustments. This article can only consider that when the price of the dynamic instance fluctuates, the instance is terminated when the price rises above the user's highest accepted bid. When the price drops, if the cloud provider's pricing is lower than our bidding instance, it will still run. As shown in Fig. 4.

In Fig. 4, we can see how the price changes in the dynamic price instance and how the user's bid affects the bidding instance application. The blue line in the figure represents the price change value of the dynamic pricing instance in the future time period, the green line represents the user's bid price, and the abscissa is the application time of the instance, that is, the running time of the instance. Amazon's Spot instance billing has a feature. When Amazon EC2 actively terminates the user's Spot instance, the user has the opportunity to waive all or part of the fee. This is also a big difference between Spot instances and on-demand instances. Amazon EC2 has launched the finest example of billing per second. The new billing method of billing per second further helps users reduce the cost of using Spot instances. Spot instances that meet the per-second billing conditions: During the first hour of launch, if the instance is terminated by Amazon EC2 for price reasons, no fees will be incurred; if an interruption occurs after more than 1 h, it will be calculated in seconds and make it more accurate to the actual usage time; if the user actively terminates the Spot instance, even if it is less than 1 h, it will be charged according to the actual usage time in seconds. Spot instances billed by the hour: During the first hour of launch, if the instance is terminated by Amazon EC2 due to price, there will be no cost; if the Spot instance price exceeds the user's bid, the Spot instance will be in the instance hour In the middle of the interruption, the user does not need to pay for the interrupted part of less than 1 h; if the user actively terminates the Spot instance in the middle of the instance hour, he needs to pay for the hour [20].

In Fig. 4(a), the user's bid is much higher than the highest price of the instance in 10 h. It can be intuitively known that this is not an optimal bidding strategy. But the advantage of this bidding strategy is that the user can keep the bid instance applied to keep running unless the cloud management system forcibly withdraws the bid instance. In Fig. 4(b), the user's bid for the first 3 h is higher than the price of the cloud bidding instance, so the instance is executed. During the period of 3 h to 5 h, the bidding instance of the user becomes inexecutable because the price of the instance rises above the user's highest bid.

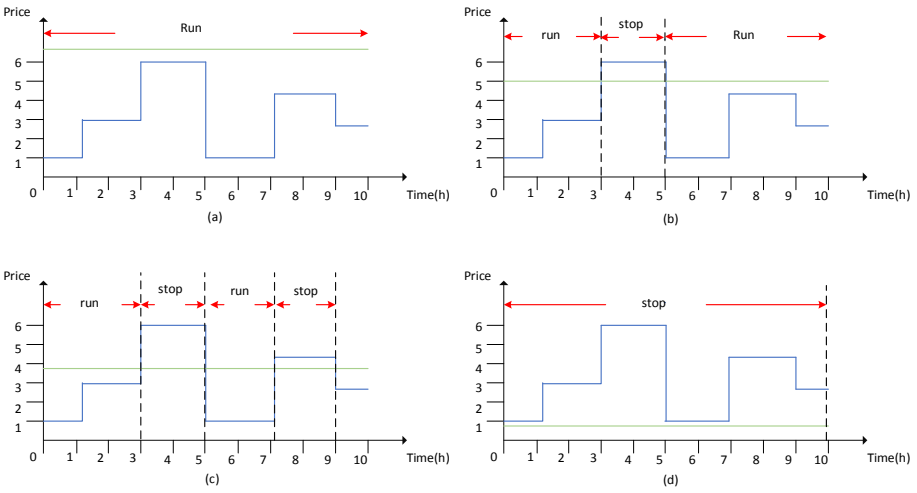

Fig. 4. The bidding price of a dynamic pricing instance (Color figure online)

Between 5 h and 10 h, the price drop instance of the bidding instance becomes feasible. The bidding strategies proposed by different cloud providers are slightly different here. After the user's bidding instance is terminated, some cloud service providers will restart the user's bidding instance when the price reaches the requirements, while some cloud service providers will directly terminate the user. The instance is running and takes back the bidding instance. The situation in Fig. 4(c) is similar to that in Fig. 4(b). The difference is that the maximum bid price of the user becomes lower, and the highest bid price is lower than the bid instance price during the application period. In Fig. 4(d), since the user's bid is always lower than the actual price of the bidding instance, the user has never been used by the instance, and the auction process fails.

For dynamic pricing instances, users need to consider not only the combination of rented instance types but also auction bidding. The auction price largely determines the time and economic benefits of task completion. Generally, when the user gives a higher bid, more computing resources can be obtained, so the task can be completed in a shorter time. However, it also needs to pay more usage fees. On the contrary, if the bidding price of the user is too low, only fewer computing resources can be obtained, and the task completion time will become longer.

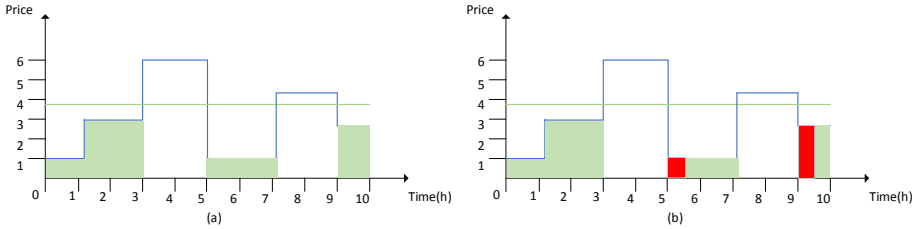

Fig. 5. The restart cost of dynamic pricing instances

While Fig. 5 shows the price of the dynamic pricing model, users bid relationship with computing resources and the cost of actually obtained. Figure 5(a) shows that when other users' bids are higher than the price of the instance without considering other factors, the instance runs. When the price of the instance changes dynamically and is higher than the user's bid, the instance is stopped. When the price of the instance is lower than the user's bid again, there is no additional cost for the instance to re-run. In Fig. 5 (a), it is executed at time [0, 3], [5, 7], [9, 10]. Therefore, the total computing power obtained by the user is an instance unit of 5 computing units. The cost of the instance in 10 h $p = \sum pi * ti$. P_i represents a stable price at which the instance is run, and t_i represents the length of time the instance is running at the price p_i. The cost P of the example in Fig. 5(a) is 9, the computing resources obtained by the job in this period of time are $3 + 2 + 1 = 6$ computing units, and the task completion time is T = W/6C. Figure 5(a) is an ideal environment. In a real environment, after an instance is restarted, initialization, job allocation, and data transmission are collectively called the restart cost. For the convenience of this article, the cost is set to 0.5-time units. Figure 5(b) is the case when the cost calculating is restarted. At this time, the cost P of the instance is still 9, the actual computing resources obtained are $3 + 1.5 + 0.5 = 5$ computing units, and

the task completion time is T = W/5C. Figure 5 considers bidding for a single instance. If bidding for multiple instances is considered at the same time, the problem becomes more complicated.

The solid color lines in Fig. 6 are actual price data of three type instances in 10 h. The yellow line indicates the instance with three VCPU units and is named as instance 3. The red line indicates the instance with two VCPU units and is named as instance 2. The blue line indicates the instance with one VCPU units and is named as instance 1. C_i represents the *i*-th computing power of the instance. The dotted lines represent the bid price for three instances under the current price bidding strategy. In Fig. 6(a) and Fig. 6(b), the computing resources obtained computing resources of 30 units in 10 h. The cost in Fig. 6(a) is 41.5, and the cost in Fig. 6(b) is 45.5. According to different bidding strategies, the three instances are bid to obtain the same computing resources, and the same workload can be processed within 10 h. The costs of users under the two bidding strategies are indeed very different. In Figs. 6(c) and 6 (d), the computing resource for user's tasks is 30 computing units. According to the difference between two bidding strategies of (c) (d), the bidding strategy in Fig. 6(c) takes 5 h to complete the task while the bidding strategy in Fig. 6(d) takes 7 h to complete the task. So, it is clear that different bidding strategies will affect the cost and completion time of the task.

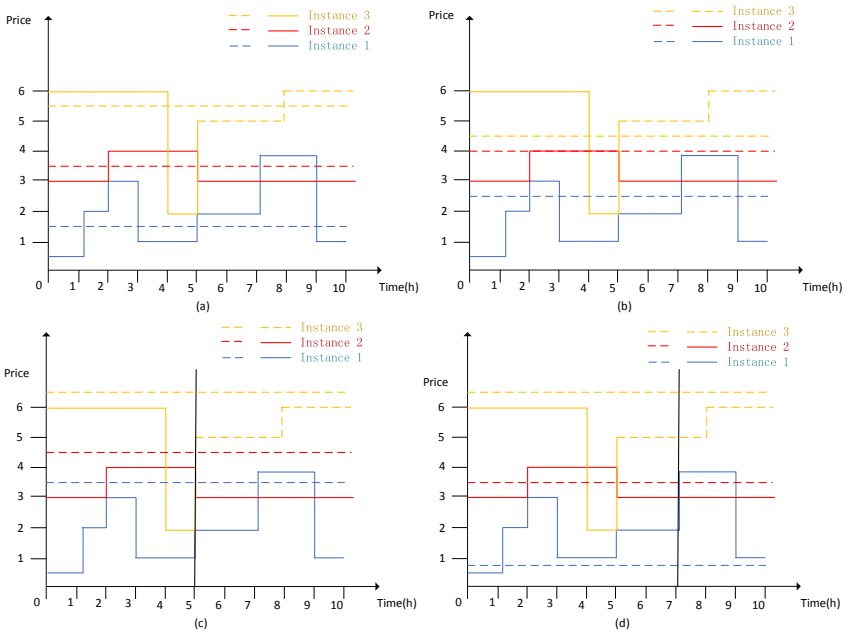

Fig. 6. Comparison of different bidding strategies (Color figure online)

4.2 Dynamic Bidding Model

Users choose IaaS instances to run their jobs. In the research questions of this article, they believe that users are pursuing the economic benefits and quality of tasks. Therefore, users will try to choose a better use strategy to maximize their economic benefits. It is assumed here that the tasks submitted by users are easily segmental, parallelizable, and computationally intensive, such as graphics rendering, video encoding, and machine learning. The total workload of the task is expressed as Workload, simply written as W. W represents the time required for each job to run and complete on an instance of unit performance. Assuming that the performance of the instance of the unit CPU is C, ideally, the completion time of the task using a single CPU is T = W/C. When a user rents multiple instances to form a combination of instances, the parallel CPU calculation unit is n. For example, if the user rents a single CPU instance and a 2CPU instance at the same time, the parallel CPU unit is 3. According to the above assumptions about tasks, tasks are splittable and easy to parallel, so the actual running time of the task is $T_{real} = W/(nC)$, then $T_{real} \leq T$. As you can see, users can rent more instances to complete tasks faster. However, it also needs to pay more for instance rental costs. Therefore, it is necessary to balance the time and economic benefits of renting instances. The optimization of user costs can be expressed as:

$$u^* = \underset{u \in U}{argmin}\, Expenses \tag{1}$$

Expenses are the user's cost of renting an instance, and u* represents the bidding price of the instance when the cost is minimized. At present, the research on dynamic pricing is generally an example of a bidding strategy [8]. This article considers the bidding problem of a combination of multiple dynamic pricing instances. The minimum cost-optimized in this paper is the sum of the costs of combining multiple instances.

$$Expenses = \sum\nolimits_1^n Expenses_i$$

The cost of each instance in actual usage time is

$$Expenses_i = \sum\nolimits_1^T f(price_i) * y_i * t_i$$

Among them, $price_i$ is the price of the instance in the current period. In general, $price_i$ is a set of price values that contain changes in time sequence, where t_i is the running time of the instance in a running cycle, that is, the pricing time of the instance, and there is a mapping relationship with the price $price_i$. y_i is whether the instance will be used within the current time, y_i is determined by instance pricing and pricing, y_i is 0 or 1. If y_i is 0, it means that the instance has not been used, and y_i means that the instance is running and starts charging.

$$Workload = \sum\nolimits_1^n \sum\nolimits_1^T \mathrm{Workload}_{ij},$$

We use *res* to represent whether the instance is in a restarted state or no after being stopped. *res* is 0 to indicate that the instance is newly started. *res* is 1 to indicate that the

instance is to be restarted. It takes *restart_cost* hours to restore the job site. Therefore, no job was run within *restart_cost* hours. The optimization problem equation is:

$$b^* = \underset{u \in U}{argmin} \sum_1^n \sum_1^T f\left(price_{ij}\right) * y_{ij} * t_{ij} \tag{2}$$

Subject to

$$Workload = \sum_1^n \sum_1^T \text{Capacity}_{ij} * (t_{ij} - \text{restart\_cost} * res_{ij}) * y_{ij}$$

In the research problem of this paper, users submit resource requests for Workload calculation and bid on a group of instances through bidding, that is, $b = [b_1, b_2, \ldots, b_n]$, where n is the type of instance. b^* is the optimal solution to be sought. The restrictions indicate that the workload requested by the user must be satisfied.

5 Iterative Optimization Algorithm for Dynamic Pricing

How users can bid to minimize revenue is the issue to be studied in this section. In our problem, we do not seek the optimal solution of the strategy but obtain the approximate optimal solution of the problem through a heuristic algorithm. Under the condition of meeting the user's needs, obtaining the approximate optimal solution can greatly improve the efficiency of the system and reduce the user's expense. We mainly use maximum price bidding algorithms, differential evolutionary algorithms, and adaptive differential evolution algorithm. The maximum price bidding algorithms are intuitive and easy to understand. Differential evolutionary is to imitate the natural evolution of the solution of our target constant optimization, know to meet the conditions of the problem solution. Our adaptive differential evolution algorithm optimizes the parameters of the basic evolution algorithm to meet the complex nonlinear data problems. We will explain the algorithms below.

5.1 The Highest Bidding Auction Strategy

First, we introduce a heuristic bidding strategy. The heuristic algorithm is easy to implement and has achieved good results in work. It is called the maximum price bidding strategy. When using the highest price auction strategy, users consider the possible price range of the bidding instance and always bid the highest price. In the maximum price bidding strategy, because users always bid at the highest possible price, the bidding instance will never be terminated by the cloud service provider. In other words, the maximum price bidding strategy can ensure the consistent execution of any job. Therefore, the time to complete the task is often minimized. Secondly, assuming that there is an optimal strategy in the bidding strategy, if the strategy of bidding at the highest price is not much different from the actual benefit of this optimal strategy, then the bidding at the maximum price is also similar to the optimized bidding strategy. This is more dependent on the price data distribution of the bidding instance, for example, the price change range of the bidding instance is relatively small.

When using the bidding strategy with the maximum price, as shown in Fig. 7, the user doesn't need to consider the change of instance price. They only need to know the maximum value of the bidding instance's price within a specified time and bid at this maximum value. In the maximum price bidding model, since the instance keeps running once applied, the job can generally be completed in the shortest time.

Fig. 7. The highest bidding auction strategy

The cost the user needs to pay is

$$Cost = \sum_{i=1}^{n} Price_i * t_i$$

$Price_i$ in the formula represents the stable bid price for a period of time, and t_i represents the length of time that the price of the bid instance remains at $Price_i$. If the running time of the task is T, then there is

$$T = \sum_{i=1}^{n} t_i.$$

5.2 Heuristic Global Optimization Algorithm Differential Evolution Algorithm

The basic idea of the algorithm is to start with a randomly generated initial population and generate a new individual by summing the vector difference of any two individuals in the population with the third individual, and then combine the new individual with the corresponding individual in the contemporary population In comparison, if the fitness of the new individual is better than the fitness of the current individual, the old individual will be replaced with the new individual in the next generation, otherwise, the old individual will be preserved. Through continuous evolution, retain good individuals, eliminate inferior individuals, and guide the search towards the optimal solution. The algorithm steps are shown in Fig. 8.

Step 1: Initialize. N_P population individuals need to be randomly and uniformly generated in the solution space. The first thing to determine is the dimension D of each individual. X_{Low} and X_{High} represent the upper and lower limit vectors of the D-dimensional space, respectively. Generate the i^{th} individual of the population: $x_i^0 = X_{Low} + \text{random}(0, 1) * (X_{High}\text{-}X_{Low})$, $i \in [1, 2, \ldots, N_P]$. The initial population of N_P individuals $X^0 = [x_1^0, x_2^0, \ldots, x_{NP}^0]$ determine the variation factor F and crossover CR,

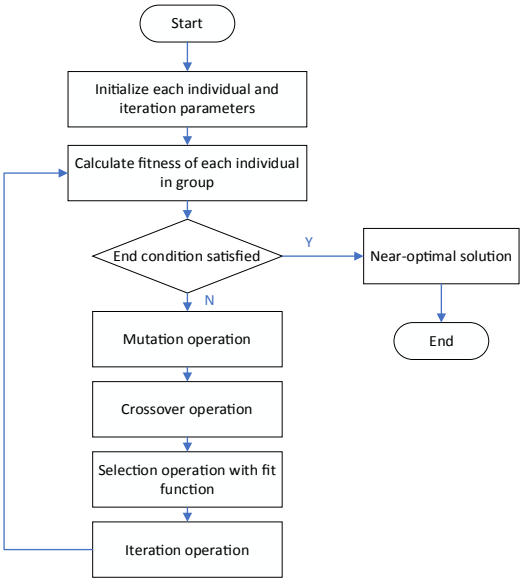

Fig. 8. Differential evolution procession

determine the maximum algebra G_{max}, the number of individuals N_P is generally greater than 4.

Step 2: For individuals in the G^{th} x_i^G ($i = 1, 2, \ldots, Np$), calculate the fitness value PE (x_i^G) of each individual, and calculate the best individual with the best fitness.

Step 3: Mutation. For the i^{th} individual in the G^{th} belt, three different individuals in the G^{th} band except with x_i^G which is $x_{r1}^G, x_{r2}^G, x_{r3}^G$ are selected, r1, r2, r3, $i \in [0, N_P]$ and $r1 \neq r2 \neq r3 \neq i$, Generate new mutant individuals $v_i^{G+1} = x_{r1}^G + F\ (x_{r2}^G\text{-}x_{r3}^G)$, generate mutant population $v^{G+1} = [v_1^{G+1}, v_2^{G+1}, \ldots, v_{NP}^{G+1}]$. $F \in [0, 2]$.

Step 4: Intersection. Cross the original individual x_i^G and the mutated individual v_i^{G+1} to get the cross individual u_i^{G+1}, $CR \in [0, 1]$ is the cross probability, r = random (0, 1) is the average distribution from 0 to 1. Cross operation can maintain the diversity of the population.

$$u_{ij}^{G+1} = \begin{cases} v_{ij}^{G+1}, & r_{ij} \leq CR \\ x_{ij}^G, & else \end{cases} \tag{3}$$

$j \in [0, D]$, it represents a dimension of each individual. Generally speaking, the greater the value of CR, the faster the convergence rate, but the convergence rate will decrease after exceeding a certain value, and the CR value will tend to premature.

Step 5: Selection is to determine whether the mutated individual u_i^{G+1} will become a new individual in the next generation. According to the greedy strategy, choose the x_i^{G+1} with better fitness of x_i^G and u_i^{G+1} as the G + 1 generation.

Step 6: Repeat steps 2–5 until the end condition of the iteration is reached.

5.3 Improved Differential Evolution Algorithm

The differential evolution algorithm has been widely used and developed due to its simplicity and efficiency. However, the algorithm itself has obvious deficiencies. The key step variation of the differential evolution algorithm is to use the difference information of the population to correct the individual information. However, as the number of iterations increases, under the action of the selection operation, the individual differences continue to decrease, so that the convergence rate becomes slower in the later iterations, and it is easy to fall into the local optimum and form premature phenomenon. In order to improve the convergence speed of the algorithm and overcome the premature algorithm, many scholars have proposed an improved algorithm for the differential evolution algorithm.

5.3.1 Improvement of Differential Strategy

Several differential strategies are listed in Table 1. In DE/x/y, x represents the base individual vector, the value of x is rand random individual, best optimal individual, current optimal individual. y represents the number of different vectors. x_{best}^{G} is the i[th] generation optimal individual, x_i^G is the i[th] generation current individual.

Table 1. Differential strategies

Differential strategy	Expression
DE/rand/1	$v_i^{G+1} = x_{r1}^G + F * \left(x_{r2}^G - x_{r3}^G\right)$
DE/rand/2	$v_i^{G+1} = x_{r1}^G + F * \left(x_{r2}^G + x_{r3}^G - x_{r4}^G - x_{r5}^G\right)$
DE/best/1	$v_i^{G+1} = x_{best}^G + F * \left(x_{r2}^G - x_{r3}^G\right)$
DE/best/2	$v_i^{G+1} = x_{r1}^G + F * \left(x_{r2}^G + x_{r3}^G - x_{r4}^G - x_{r5}^G\right)$
DE/randtobest/1	$v_i^{G+1} = x_i^G + F * \left(x_{best}^G - x_i^G\right) + F * \left(x_{r1}^G - x_{r2}^G\right)$

DE/rand/1/ and DE/best/2/ are currently the most widely used and most successfully applied differential strategies. DE/rand/1/ is conducive to maintaining the diversity of groups, and the DE/best/2/ strategy is more concentrated on the convergence speed of the algorithm. In our improved differential strategy, this article uses a combination strategy of DE/rand/1/ and DE/best/2. The parameter λ_{rand} represents the probability of the algorithm adopting the DE/rand/1 differential strategy, $\lambda_{\text{rand}} \in [0, 1]$. At the beginning of the iteration, the algorithm uses more DE/rand/1, which can maintain the diversity of races and prevent falling into the local optimal solution. In the later stage of the iteration, DE/best/2 is used more so that the algorithm can converge to the optimal solution faster. The differential strategy is as follows:

$$v_i^{G+1} = \begin{cases} x_{r1}^G + F * \left(x_{r2}^G - x_{r3}^G\right), & random[0, 1] < \lambda_{rand} \\ x_{r1}^G + F * \left(x_{r2}^G + x_{r3}^G - x_{r4}^G - x_{r5}^G\right), & else \end{cases}$$

5.3.2 Adaptive Mutation Factor

The mutation factor F is an important parameter of the differential evolution algorithm that controls the mutation of individuals in the population. Generally, the greater the F, the greater the impact on individual variation, which is conducive to maintaining the diversity of the population. However, the larger the F value, the greater the fluctuations in the mutant individuals, which will reduce the search efficiency and the accuracy of finding the global optimal solution. When the F value is smaller, the impact on individual variation is smaller, which can serve the purpose of searching for the optimal solution locally, but it is easy to make the algorithm premature. We improve the linear method presented in paper [21] by adopting a linear method of adjusting the variation factor F. Through experiments, it is known that too small or too large F generally improves the algorithm less. When the value of F is 0.4 to 1, it tends to have better results. Introduce parameters here $\xi = random\left[\left(\frac{G_{max}-G_i}{G_{max}}\right)^2, \frac{G_{max}-G_i}{G_{max}}\right]$. L represents the length of the interval that changes randomly, and the changing trend of L is $0 \rightarrow 1/4 \rightarrow 0$. The adaptive mutation factor in this paper is expressed as follows:

$$F_x = \xi * F_{max} + (1 - \xi) * F_{min} \tag{5}$$

$F_i = \max(F_x, F_{min})$ In this article, there is a tendency for the mutation to become smaller gradually so that in the early stage of the iteration, there is a high probability that *Fmax*, and in the later stage of the iteration, the value of F has a greater probability of approaching *Fmin.* The differential evolution algorithm can maintain the population diversity in the early stage, which prevents prematurely, and may accurately locate the local optimal solution in the later stage. Not only maintains the trend of F from large to small but also makes the variation factor have certain randomness to better maintain population diversity and better find the global optimal solution.

5.3.3 Adaptive Crossover Factor

The crossover factor *CR* is another important parameter of the differential evolution algorithm. The crossover factor has an important influence on the convergence speed of the algorithm. The larger the value of *CR*, the greater the contribution of the mutant individual to the iteration, which is beneficial to the local search of the algorithm and accelerates the convergence rate of the algorithm; the smaller the value of *CR*, the greater the contribution of the current individual to the iteration, which is conducive to maintaining population diversity. Beneficial to the global search of the algorithm. A good search algorithm should maintain the diversity of the population in the early stage of the iteration and strengthen the convergence speed of the local search in the later stage. Therefore, this article uses the adaptive crossover factor to express as follows:

$$CR_x = (1 - \xi) * CR_{max} + \xi * CR_{min} \tag{6}$$

$CR_i = \max(CR_x, CR_{min})$ The difference factor in this paper has a greater probability to obtain a smaller value at the beginning of the iteration, which can well maintain the diversity of races, and has a greater probability to obtain a larger value in the later period, which allows the algorithm to quickly It converges to the local optimal solution, while the adaptive parameters maintain certain randomness.

6 Experimental Evaluation

In this section, this article downloads the actual Spot price historical data from the Amazon cloud official website platform. The experiment selects 12 types of instances and optimizes the bidding strategy for 12 instances through the adaptive parameter differential evolution algorithm.

6.1 Data Description

Downloading the actual spot historical price data from Amazon, the price data has a period of nearly 3 months from 2018.9.28 to 2018.11.23. Amazon cloud instances are classified into general-purpose instances according to optimization and usage, computing-optimized instances, memory-optimized instances, accelerated computing instances, storage-optimized instances, etc. Generally speaking, users need the same kind of instances for one job application Types of. The experiments in this paper are set to be computationally intensive. Therefore, the experiments in this paper chose the instances of computational optimization as the research object in the experiment and selected 12 types of instances of computational optimization as the types of instances of our bidding strategy optimization. The computing-optimized instance operating system is Linux/ Unix, and the usable area is us-east-1a. Table 2 lists the names and detailed parameters of the 12 computationally optimized instances of our experiment.

Table 2. Instances configuration comparison

Instance type	Number of VCPU	Memory	Instance type	Number of VCPU	Memory
c5.large	2	4	c5d.large	2	4
c5.xlarge	4	8	c5d.xlarge	4	8
c5.2xlarge	8	16	c5d.2xlarge	8	16
c5.4xlarge	16	32	c5d.4xlarge	16	32
c5.9xlarge	36	72	c5d.9xlarge	36	72
c5.18xlarge	72	144	c5d.18xlarge	72	144

Figure 9 shows the price data for 12 instances. Due to the large difference in prices of different instances, to better show the fluctuation of the output price of the instance, we show the prices of the instances in four sub-graphs. It can be seen from the figure that the price of the instance fluctuates greatly.

6.2 Settings of Experimental Parameter

This article conducts strategy optimization experiments on the real data set of Amazon Cloud Spot instances. The experiment uses a differential evolution algorithm to optimize the bidding strategy of the dynamically priced IaaS instance. The parameters of the differential evolution algorithm are the minimum value of the F mutation factor 0.4, the

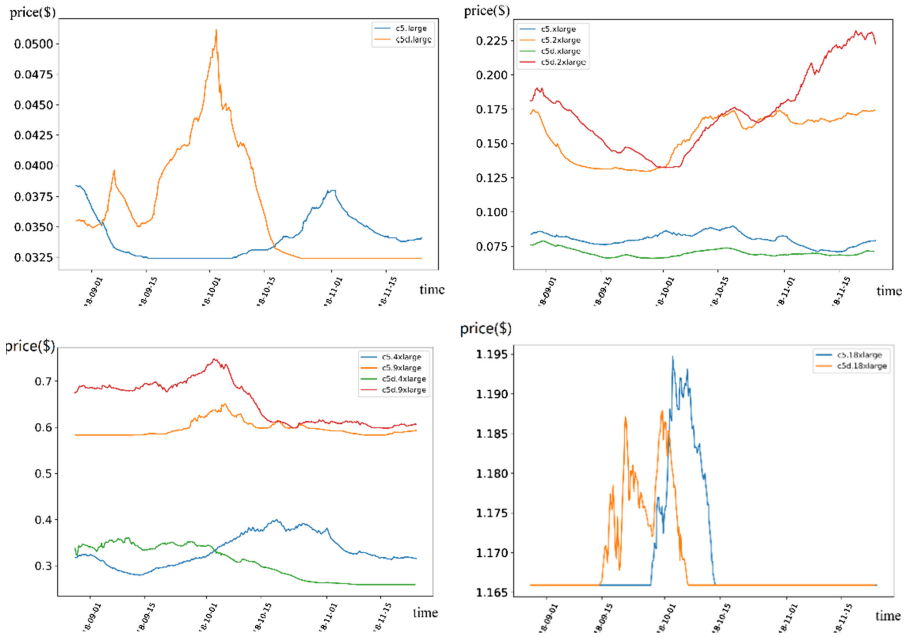

Fig. 9. Comparison of instances price data

maximum value 1, the minimum value of the *CR* crossover factor 0.3, the maximum value 0.9, the data dimension is the number of instances, and the number of populations 15. The number of iterations is 200. First, the workload is used as the task's limiting condition, and the vector value of the individual in the population is used to calculate the time required to meet the current workload, and then the time is calculated to obtain the fitness function value of the optimization problem. In the experiment, a variety of Workload values were entered, and our optimization method was tested under different conditions of demand, and the optimal bidding strategy under each workload, and the task completion time and total cost under the current optimal bidding strategy were obtained.

6.3 Experiment Results and Analysis

Table 3 shows the optimization results of the bidding strategy through the adaptive differential evolution algorithm. H represents hour and D represents the day. The workload is the demand of the user task, that is, the running time of the CPU with the unit performance required by the task. The strategy combination is the optimal bidding combination found by the differential evolution algorithm under each workload. The completion time represents the time required to complete the current workload under the current strategy, and finally the total cost. Through the differential evolution algorithm, the minimum cost value of the task whose workload satisfies is found.

Table 4 compares the performance of the algorithm. In the strategy optimization for handling different workloads, the adaptive differential evolution algorithm has a

Table 3. Optimal bidding strategy

Workload	Optimization strategy	Time	Cost
200	0.0334, 0.0714, 0.1567, 0.2783,0.5677, 1.0666, 0.0407, 0.0640, 0.1762, 0.3069, 0.6028, 1.0622	4 h	3.5848
500	0.0301, 0.0858, 0.1249, 0.3197, 0.0585, 1.0727, 0.0382, 0.0638, 0.1686, 0.2763, 0.6509, 1.0612	11 h	8.4360
1000	0.0335, 0.0700, 0.1553, 0.2766, 0.5836, 1.0567, 0.0297, 0.0686, 0.1656, 0.3188, 0.6379, 1.0684	1d 3d	16.3240
5000	0.0343, 0.0797, 0.1490, 0.2832, 0.5917, 1.0478, 0.0310, 0.0665, 0.1160, 0.2434, 0.5945, 1.0500	5d 18 h	80.4540
10000	0.0308, 0.0632, 0.1347, 0.2758, 0.5840, 1.0649, 0.0313, 0.0676, 0.1640, 0.2557, 0.5472, 1.0678	11d 13 h	162.0740
30000	0.0335, 0.0744, 0.1570, 0.2543, 0.5904, 1.0572, 0.0362, 0.0677, 0.988, 0.2692, 0.4303, 1.0510	32d 13 h	491.9586
50000	0.0340, 0.0733, 0.1487, 0.2499, 0.5863, 1.0610, 0.0301, 0.0589, 0.1533, 0.2724, 0.5946, 1.0562	82d 18 h	823.4141

very good performance improvement. Meanwhile, when the workload increases, the preferential ratio maintains a relatively stable price, which shows that the adaptive differential evolution algorithm in this paper has a good optimization effect on short-term task requirements and long-term task requirements. The adaptive differential evolution algorithm has an average cost optimization of 12.48% compared to the maximum bidding strategy, and an average cost optimization ratio of 3.53% over the basic differential evolution algorithm.

Table 4. Comparison of optimization results

	Maximum bid	Differential evolution	Adaptive differential evolution	Better than the biggest bid	Special differential evolution discount
200	4.8370	3.7116	3.5848	25.98%	3.41%
500	9.6741	8.6415	8.1620	15.60%	5.54%
1000	19.3486	17.4566	16.3420	15.63%	6.38%
5000	90.9054	82.6310	80.4540	11.50%	2.63%
10000	179.4283	165.6529	162.0740	9.36%	2.16%
30000	529.1734	504.3787	491.9586	4.88%	2.46%
50000	880.0985	841.2941	823.4141	4.41%	2.13%

7 Conclusion

This paper introduces the dynamic pricing cloud service instance types and introduces the characteristics of cloud dynamic pricing instance types. Then explained the analysis of the bidding strategy of the dynamic pricing cloud service instance and compared it with the fixed price instance. Then, the optimization strategy of the dynamically priced cloud instance is modeled, and the expression for optimizing the user's job cost is obtained. Then it introduces the maximum bidding strategy and the bidding strategy based on the differential evolution algorithm of the global optimization algorithm. In the problem of this paper, the model can simultaneously optimize the bidding and usage strategies of multiple instances. Finally, download the real price history data of the Amazon cloud dynamic instance Spot instance, and calculate the performance parameters of the optimized instance in reference 12, and use the adaptive differential evolution algorithm to perform dynamic pricing instances when users use the combination of instances. Strategy optimization to achieve the purpose of reducing costs. The experimental results show that the adaptive differential evolution algorithm has a good optimization effect for both short-term task requirements and long-term task requirements. The adaptive differential evolution algorithm has an average cost optimization of 12.48% compared to the maximum bidding strategy, and an average cost optimization ratio of 3.53% over the basic differential evolution algorithm.

In this paper, the types of jobs are limited and only consider parallel tasks. In a real environment, users' tasks are complex and diverse. So, how the bidding strategy is suitable for various task types is the problem that needs to be considered.

Acknowledgement. The authors would like to acknowledge the support provided by the National Key R&D Program of China (2018YFB1404501), and the Young Scholars Program of Shandong University.

References

1. Amazon Cloud EC2 (2020). https://aws.amazon.com/ec2/
2. Google Compute Engine (2020). https://cloud.google.com/compute/
3. Yang, S., Pan, L., Wang, Q., et al.: Subscription or pay-as-you-go: optimally purchasing IaaS instances in public clouds. In: International Conference on Web Services, pp. 219–226 (2018)
4. Amazon Cloud Spot instance (2020). https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html
5. Wolski, R., Brevik, J.: Providing statistical reliability guarantees in the AWS spot tier. In: High Performance Computing Symposium (2016)
6. Vecchiola, C., Pandey, S., Buyya, R., et al.: High-performance cloud computing: a view of scientific applications. In: International Symposium on Pervasive Systems, Algorithms, and Networks, pp. 4–16 (2009)
7. Karunakaran, S., Krishnaswamy, V., Sundarraj, R.P.: Decisions, models and opportunities in cloud computing economics: a review of research on pricing and markets. In: Australian Symposium on Service Research and Innovation (2014)
8. Tang, S., Yuan, J., Li, X., et al.: Towards Optimal bidding strategy for Amazon EC2 cloud spot instance. In: International Conference on Cloud Computing, pp. 91–98 (2012)

9. Benyehuda, O.A., Benyehuda, M., Schuster, A., et al.: Deconstructing Amazon EC2 spot instance pricing. Electron. Commer. **1**(3), 1–20 (2013)
10. Kaminski, B., Szufel, P.: On optimization of simulation execution on Amazon EC2 spot market. Simul. Model. Pract. Theory **58**, 172–187 (2015)
11. Bharad, V.H., Bheda, H.A.: SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter. Int. J. Comput. Appl. (0975 – 8887) **112**(16) (2015)
12. Zhu, J., Hong, J., Hughes, John G.: Using Markov chains for link prediction in adaptive web sites. In: Bustard, D., Liu, W., Sterritt, R. (eds.) Soft-Ware 2002. LNCS, vol. 2311, pp. 60–73. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46019-5_5
13. Zheng, B., Pan, L., Yuan, D., Liu, S., Shi, Y., Wang, L.: A truthful mechanism for optimally purchasing IaaS instances and scheduling parallel jobs in service clouds. In: Pahl, C., Vukovic, M., Yin, J., Yu, Q. (eds.) ICSOC 2018. LNCS, vol. 11236, pp. 651–659. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03596-9_47
14. Shao, Q., Liu, S., Pan, L., Yang, C., Niu, T.: A market-oriented heuristic algorithm for scheduling parallel applications in big data service platform. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, pp. 677–686 (2018)
15. Storn, R., Price, K.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**(4), 341–359 (1997)
16. Kim, H., Chong, J., Park, K., et al.: Differential evolution strategy for constrained global optimization and application to practical engineering problems. In: IEEE Conference on Electromagnetic Field Computation, vol. 43, no. 4, pp. 1565–1568 (2006)
17. Omran, M.G., Engelbrecht, A.P.: Self-adaptive differential evolution methods for unsupervised image classification. In: IEEE Conference on Cybernetics and Intelligent Systems, pp. 1–6 (2006)
18. Zhang, R., Ding, J.: Non-linear optimal control of manufacturing system based on modified differential evolution. In: IMACS Multiconference on Computational Engineering in Systems Applications. IEEE (2006)
19. Onwubolu, G.C., Babu, B.V.: New Optimization Techniques in Engineering. Springer, Berlin Heidelberg (2004). https://doi.org/10.1007/978-3-540-39930-8
20. Amazon spot instance detail. https://aws.amazon.com/cn/blogs/china/amazon-ec2-spot-instance-detail/
21. Liang-Hong, W., Hui-Nan, W.: Adaptive quadratic differential evolution algorithm. Control Decis. **21**(8), 898–902 (2006)