Optimal Static Bidding Strategy for Running Jobs with Hard Deadline Constraints on Spot Instances

Kai-Siang Wang, Cheng-Han Hsieh and Jerry Chou
National Tsing Hua University, Hsinchu, Taiwan, Republic of China

Keywords: Cloud Computing, Bidding Strategy, EC2, Spot Instance, Deadline Constraint.

Abstract: *Spot-instances*(SI) is an auction-based pricing scheme used by cloud providers. It allows users to place bids for spare computing instances and rent them at a substantially lower price compared to the fixed on-demand price. This inexpensive computational power is at the cost of availability, because a spot instance can be revoked whenever the spot market price exceeds the bid. Therefore, SI has become an attractive option for applications without requiring real-time availability constraints, such as the batch jobs in different application domains, including big data analytics, scientific computing, and deep learning. For batch jobs, service interruptions and execution delays can be tolerated as long as their service quality is gauged by an execution deadline. Hence, this paper aims to develop a static bidding strategy for minimizing the monetary cost of a batch job with hard deadline constraints. We formulate the problem as a Markov chain process and use Dynamic Programming to find the optimal bid in polynomial time. Experiments conducted on real workloads from Amazon Spot Instance historical prices show that our proposed strategy successfully outperformed two state-of-art dynamic bidding strategies (*Amazing, DBA*), and several deadline agnostic static bidding strategies with lower cost.

1 INTRODUCTION

Clouds have become an attractive computing platform for many applications. It is a computing paradigm to deliver ready-to-use, *on-the-fly* configurable resources as services on Internet, and to charge these services on a *pay-as-you-use* pricing model. The elasticity and on-demand characteristics of cloud computing further ensure higher resource utilization and lower computing cost to users. One of the most famous cloud services is Amazon's EC2 (EC2, 2009). EC2 provides raw computing resources in the form of Virtual Machines(VM). To meet computing requirements of a wide variety of applications, different pricing options are offered, including ***On-Demand*** **(OD)**, ***Reserved*** and ***Spot Instances*** **(SI)**. On-Demand provides dedicated access to a set of machines for a fixed cost per hour with no long term commitments. On the other hand, Reserved offers a significant discount on hourly charge over On-demand, but users must make a one-time, upfront payment to lease the VMs for long periods of time (1 or 3 year terms).

In contrast to On-Demand and Reserved instances which are charged with fix price rates, Spot Instances allow customers to bid on spare computing capacity with no upfront commitment and at a variable hourly rates called the **spot price** (or market price). The spot price is determined by the cloud providers based on the demand of VMs within their infrastructure. If the **bid price** submitted by a user is higher than the spot price (which is called **in-bid**), then the user receives his requested instances and only pay at the cost of spot price. Hence, many studies (Yi et al., 2012a; Mazzucco and Dumas, 2011) have shown that spot instances can greatly reduce the execution cost. But, this inexpensive computational power is at the cost of availability, because a spot instance can be revoked whenever the spot market price exceeds the bid (which is called **out-bid**). Furthermore, when a out-bid event occurs, not only the VM becomes unavailable for the given hour, its jobs also need to be recovered from a previous checkpoint. As a result, a lower bid price can reduce the monetary cost, but also potentially increase the job execution time.

While SI may not meet the computing requirements of any application, it is an attractive option for batch jobs. A batch job is often described by a computation time and a deadline, The computation time can be spread within the time-window specified by the deadline, so that the service interruptions and execution delays caused on the revocable resource can be tolerated. (Andrzejak et al., 2010a) is one of the earli-

Wang, K., Hsieh, C. and Chou, J.
Optimal Static Bidding Strategy for Running Jobs with Hard Deadline Constraints on Spot Instances.
DOI: 10.5220/0011645400003488
In *Proceedings of the 13th International Conference on Cloud Computing and Services Science (CLOSER 2023)*, pages 123-130
ISBN: 978-989-758-650-7; ISSN: 2184-5042

est work to show the potential cost saving from using SI, and analyze the complicated correlation between bid price, job execution time and monetary cost on SI. Since then, many research efforts have been made aiming to find the bidding strategies for minimizing monetary cost under different problem settings, including the constraints of job execution, the checkpoint scheme for fault recovery, the transition model of spot price, etc.

Amazing (Tang et al., 2012) and DBA(dynamic bidding algorithm) (Song et al., 2012) are the two start-of-art dynamic approaches that claims to achieve optimal bids for job with deadline constraints. *Amazing* uses a dual-option bidding strategy, which either bids on the max price or zero dollars for the next instance hour according to the current running state and spot price. It formulates this problem as a Constrained Markov Decision Process (CMDP), and finds an optimal randomized bidding strategy through linear programming. However, *Amazing* only guarantees the *expected* execution time to be less or equal to the deadline, so it is an approach for job with soft deadline constraints. On the other hand, *DBA* guarantee jobs to be finished before their deadlines by switching to On-Demand instances when necessary. Like most dynamic approaches, it assumes the bid price can be adjusted without interrupting the VM instances. But in the real EC2 environment, spot instance request can only be associated with one bid price. Thus, in order to change bid price, users have to cancel the existing SI request, and then re-create a new request with the new bid. As a result, in practice, users still prefer to use static bidding strategies which use a fixed bid price throughout the job execution.

To accommodate to the real cloud provider environment and cloud users behavior, this work proposed a static bidding strategy for jobs with hard deadline constraint and using hourly checkpoint recovery. This is one of the most commonly seen use scenarios for spot instance. But, to our knowledge, its optimal bids have not been discussed in the previous literature. We formulate the problem as a finite-time stochastic dynamic program, and prove the optimal bid can be determined in polynomial time. We also show our formulation can be extended to include the execution time overhead from checkpoint and restart process. The performance of our algorithm is extensively evaluated using a real data-set of Amazon EC2 spot price history obtained from (EC2,). By comparing with the state-of-art dynamic bidding strategies (*Amazing, DBA*), and several deadline agnostic static bidding strategies, we demonstrate that our optimal static strategy can achieve the lowest cost with better computation efficiency and applicability in practice.

The remainder of this paper is organized as follows: Section 2 introduces the characteristics of spot instances and our problem formulation. Our bidding algorithm is describe in Section 3 and extensively evaluated in Section 4. Finally, related work and conclusions are given in Section 5 and Section 6, respectively.

2 SPOT INSTANCES & PROBLEM FORMULATION

2.1 Spot Instance Characteristics

Amazon introduces spot instances as a bidding option for excess EC2 resources. Currently there are 64 types of Spot Instances available in market, and they differ by computing power, memory/disk space, OS and geographical location, etc (Stokely et al., 2009a). Amazon sets the **spot price**, which fluctuates depending on the supply and demand of Spot Instance capacity. While the Spot Price may change anytime, in general the spot price will change once per hour, which is named as **instance hour** (or time interval) in this work, and in many cases less frequently. As observed, the spot price is often less than 1/3 of the on-demand price. But occasionally, the spot price can also exhibit some spikes higher than the on-demand instance price. Nevertheless, in general, spot instances still provide a huge cost saving opportunity to users (Yi et al., 2012b).

To use Spot Instances, users place a **spot instance request**, specifying the instance type, the AWS region, the number of instances, and the maximum price they are willing to pay per hour which is known as the **bid price**. There are two main ways of requesting an instance also referred as bidding strategies: the dynamic bidding strategy consists of placing a bid at each time epoch while the static bidding strategy decides only one bid until completion of the job. Currently, EC2 doesn't allow users to change the bid price of a Spot Instance request once it is submitted. To change the bid price, users have to cancel the original request, stop the running instances and create a new request with the updated bid price. Thus, as shown in figure. 1, dynamic strategy requires to set a bid for each time epoch while static bidding strategy bids with the same price for the course of the execution. When a user's bid is higher than the current spot price, the requested resource is granted and the user is charged by the current spot price, not the bid price. In contrary, when a user's bid is less than the current price, the requested instance is stopped imme-

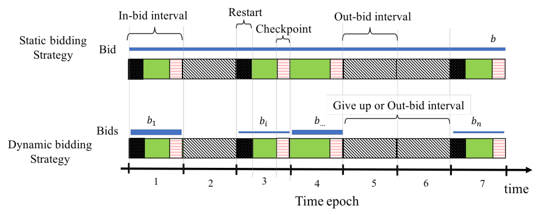

Figure 1: Dynamic and Static bidding strategies on Spot Instances. The green intervals indicates the effective job computation time. b is the static unique bid price; and for dynamic strategy, b_i represent the bid price for each time instant.

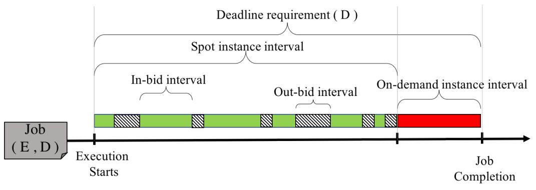

Figure 2: Spot instance Model: the in-bid intervals are the time epochs in which the resource is granted. The red intervals require an on-demand instance to complete the job. The completion time of the job includes the sum of all intervals and is bounded by the given deadline D. The sum of in-bid and on-demand intervals must be higher or equal to the estimated execution time E.

diately[1], and the instance usage of that partial hour is not charged. In this work, we call the former case as **in-bid interval**, and the later case as **out-bid interval**. To recover the job progress after out-bid intervals, users must make checkpoint (i.e. VM snapshot) at the end of an in-bid interval, and restart from the next in-bid interval using the snapshot VM image to resume the job. As shown in figure. 1, both checkpoint and restart require additional time which often depends on the resource usage of an instance (Jangjaimon and Tzeng, 2015). Thus these overhead are also considered in our problem formulation and experiments.

2.2 Spot Price Model

There is very limited information available from Amazon as to how they determine the spot price. Amazon does not reveal bids by users or the amount of demand. But the historical spot prices are made available and accessible via the Amazon Console API (Amazon, 2004). Hence, there is a general agreement in the recent literature (Song et al., 2012; Chohan et al., 2010; Tang et al., 2012) that the spot price should be modeled as a stochastic process using Markov Chain, where its current spot price depends only on the previous spot price. In this work, we use $S(t)$ to denote the spot price at an instance hour t, and adapt the same price model defined in (Song et al., 2012). Let $S(t)$ be a stochastic process following the Markov model below at time t:

$$S(t) = \begin{cases} S(t-1), & \text{with probability } p \\ S, & \text{with probability } (1-p) \end{cases} \tag{1}$$

where S is a random variable generated from a general distribution function $f(s)$. We also use $F(s)$ to denote the cumulative distribution function of spot price, such that $F(s) = Pr(S < s)$. In practice, there are only limited number of spot prices ($|S| = n$), and $f(s)$ can be an empirical distribution function derived from observed spot price traces. Thus given a current spot price, the price will remain the same for the next instance hour with probability p, and change to a new price determined by the random variable S with probability $(1-p)$.

We note that the above synchronous discrete-time model is an an approximation of an actual continuous-time spot bidding system, and the spot price could be modeled by other Markov Chain definitions (Javadi et al., 2011). But we justify the above model by showing it captures the essential variation behavior of spot price and provides a tractable mathematical model for finding the optimal bids. In the experiments, we also use the real trace of historical spot price to evaluate our algorithms and models, and show $F(s)$ can be defined as an empirical distribution function according to the historical spot prices.

2.3 Problem Formulation

Here we formally describe our problem formulation. Table 1 summarizes all the variables we use throughout the paper. As shown in Figure. 2, the input of our problem is a job described by an execution time E and a deadline D, and the goal of our algorithms is to find an optimal bid price b^*, such that the monetary cost for completing the job on spot instances is minimized.

To form the optimization function, we first introduce the following variables to describe the running status of a job associated with a given bid price b and time interval t. As described in the previous subsection, the cloud provider generates a spot price $S(t)$ at each time interval t according to Eq. 1. If the spot price $S(t)$ is less or equal to the bid price b, user's spot instance request is granted, and the requested VM instances are charged by the spot price S. Otherwise, the user does not get any VM. Hence, we use $I(t)$ to

[1]Currently Amazon offers a 2min notice prior to stoppage.

Table 1: Variables and descriptions.

	variable	description
input	E	job execution time
	D	job deadline
output	b^*	optimal bid
problem setting	S	spot prices. ($\|S\| = n$)
	p	transition probability of spot price
	$f(s)$	spot price distribution. $F(s) = Pr(S < s)$
	S_{OD}	price of on-demand instances
	γ	restart time overhead
	κ	checkpoint time overhead
status	$B(t)$	bid price at time t
	$I(t)$	in-bid status at time t
	$K(t)$	checkpoint time spent at time t
	$R(t)$	restart time spent at time t
	$e(t)$	remaining execution time at the beginning of time t
	$c(b, s_{t-1})$	progress at the current time interval with given bid price b and previous spot price s_{t-1}

indicate whether t is an in-bid interval or out-bid interval. The cost of an in-bid interval is $S(t)$, and the cost of a out-bid interval is 0.

$$I(t) = \begin{cases} 1, & \text{if } S(t) \leqq b (t \text{ is an in-bid interval}) \\ 0, & \text{otherwise} (t \text{ is a out-bid interval}) \end{cases} \quad (2)$$

In this paper, we consider a hourly checkpoint strategy is used, and assume the time for checkpointing and restarting are constant values κ and γ, respectively. We leave it as a future work to discuss the possibility of adapting our model to other checkpoint strategies, such as rising edge. Under hourly checkpoint strategy, there is a checkpoint overhead at every in-bid interval. But the restart overhead only occurs when a out-bid interval followed by an in-bid interval. Hence, at each in-bid interval t, we use variables $K(t)$ and $R(t)$ to denote the overhead time spent on checkpoint and restart as follows.

$$K(t) = \begin{cases} \kappa, & \text{if } I(t) == 1 \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

$$R(t) = \begin{cases} \gamma, & \text{if } I(t-1) == 0 \ \&\& \ I(t) == 1 \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

Accordingly, the remaining execution time at interval t can be denoted by $e(t)$ as below. The value is taken by the ceiling function because spot instances are purchased in the unit of hourly interval.

$$e(t) = \lceil E - \sum_{i=0}^{t-1} (I(i) - K(i) - R(i)) \rceil \quad (5)$$

Therefore, let (e, s_{t-1}, t) denote the state where e is the amount of computation left at the start of interval t, and s_{t-1} is the observed spot price from the previous interval $t-1$. The monetary cost for completing the reminding execution time e starting from job state (e, s_{t-1}, t) under a given static bid b is denoted as $C_b(e, s_{t-1}, t)$.

Furthermore, to guarantee job completion before deadline, on-demand instance must be used when the reminding execution time is equal to the time left. Otherwise, any out-bid interval occurs in the future will certainly cause job exceeds deadline. For instance, if deadline is $D = 8$, and the reminding execution time is 3, then spot instances must be used from $t = 5$. Therefore, we enforce a boundary condition, such that

$$C_b(e, s_{t-1}, T) = S_{OD} \cdot \lceil e \rceil, \text{ if } \lceil e \rceil \geqq T, \forall s_{t-1} \quad (6)$$

where S_{OD} is the on-demand price.

In sum, our problem can be formulated as below:

Input: (E, D), a job description.
s_{init}, the spot price observed initially.
$< p, f(s), S_{OD}, \kappa, \gamma >$, problem settings.
Output: b, the optimal bid
Objective: $C_b(E, s_{init}, 0)$ is minimized.
Subject to: Eq. 1 $\sim$ Eq. 6.

3 BIDDING STRATEGY

Our goal is to compute $C_b(E, s_{init}, 0)$ with a given bid price b. First of all, at any state $< e, s_{t-1}, t >$, assume we already know the cost of all possible future states, $C_b(e', s'_{t-1}, t')$ where $e' \leq e, s'_{t-1} \in S$, and $t' \geq t$. Thus the monetary cost of a state $< e, s_{t-1}, t >$ can be formulated as below based on the random variable of current spot price s_t at time interval t:

$$\begin{aligned} C_b(e, s_{t-1}, t) = E_{s_t}[&C_b(e, s_t, t+1) \cdot (1 - I(s_t \leqq b)) \\ &+ (s_t + C_b(e - c(b, s_{t-1}), s_t, t+1)) \cdot I(s_t \leqq b)] \end{aligned} \tag{7}$$

The first term represents the cost when the current spot price is larger than the bid price, and the second term represents the cost when it is not. In first term, the current interval is out-bid, so the cost is simply equal to the cost from $t+1$ with the same remaining computation time e. Whereas in the second term, the current interval is in-bid, so the cost is the sum of cost incurred at current time t and the accumulated cost from time $t+1$ with the remaining computation $(e - c(b, s_{t-1}))$, where $c(b, s_{t-1})$ is the progress of job execution in the in-bid interval. In this work, we use hourly checkpoint, and the restart is only required if the previous interval is a out-bid interval. Thus, the actual progress of an interval is depending on the previous spot price and bid price as below:

$$c(b, s_{t-1}) = \begin{cases} 1 - \kappa, & \text{if } s_{t-1} \leq b \\ 1 - \kappa - \gamma, & \text{otherwise} \end{cases} \tag{8}$$

Next, we replace the value of s_t in Eq 7 with the random variable S according to the spot price defined in Eq 1. After re-writing the equation, we get

$$\begin{aligned} &C_b(e, s_{t-1}, t) = p\{s_{t-1} I(s_{t-1} \leqq b) \\ &+ C_b(e - c(b, s_{t-1}), s_{t-1}, t+1) I(s_{t-1} \leqq b) \\ &+ C_b(e, s_{t-1}, t+1)(1 - I(s_{t-1} \leqq b))\} \\ &+ (1-p)\{E_S[SI(S \leqq b)] \\ &+ E_S[C_b(e - c(b, s_{t-1}), S, t+1) I(S \leqq b)] \\ &+ E_S[C_b(e, S, t+1)(1 - I(S \leqq b))]\} \end{aligned} \tag{9}$$

As shown in the above equation, the value of s_{t-1} remains the same throughout the recursion, and b is a given fixed value. Therefore, we further simplify the equation and discuss its solution in two separate cases based on the relation between s_{t-1} and b as follows.

For the case of $s_{t-1} \leqq b$: The term with $(1 - I(s_{t-1} \leqq b))$ can be eliminated from Eq. 9, and $c(b, s_{t-1})$ is equal to $(1 - \kappa)$. Therefore, the final form of our recursion equation becomes

$$\begin{aligned} &C_b(e, s_{t-1}, t) = \\ &p(s_{t-1} + C_b(e - (1-\kappa), s_{t-1}, t+1)) \\ &+ (1-p)\{E_S[SI(S \leqq b)] \\ &+ E_S[C_b(e - (1-\kappa), S, t+1) I(S \leqq b)] \\ &+ E_S[C_b(e, S, t+1)(1 - I(S \leqq b))]\} \end{aligned} \tag{10}$$

For the case of $s_{t-1} > b$: The term with $(I(s_{t-1} \leqq b))$ can be eliminated from Eq. 9, and $c(b, s_{t-1})$ is equal to $(1 - \kappa - \gamma)$. Therefore, the final form of our recursion equation becomes

$$\begin{aligned} &C_b(e, s_{t-1}, t) = pC_b(e, s_{t-1}, t+1)) \\ &+ (1-p)\{E_S[SI(S \leqq b)] \\ &+ E_S[C_b(e - (1-\kappa-\gamma), S, t+1) I(S \leqq b)] \\ &+ E_S[C_b(e, S, t+1)(1 - I(S \leqq b))]\} \end{aligned} \tag{11}$$

Clearly, in both cases, $C_b(e, s_{t-1}, t)$ can be derived from its own equation with decreasing values of e and increasing values of t. Therefore, with the known boundary condition stated in Eq. 6, we can compute $C_b(e, s_{t-1}, t)$ recursively using a DP algorithm introduced in the next subsection.

4 EVALUATIONS

4.1 Setup

In order to evaluate the performance of our model, we collected historical prices form Amazon EC2 API (Amazon, 2004) for the period of January ∼ June 2015. For the following experiment, we consider the prices of the instance type *us-west-1a.linux-m1.small*. We derived the price transition matrix from the real traces, then mapped it to the random variable S described in the problem definition in sect. 2.2. The variation of the price interval spans from 1¢ to 1\$. By default, we set the checkpoint time to 5*min* and the required time to restart an instance to 10*min*. We compare our strategy labeled as **static** to a number of existing strategies: **Dynamic** represents the dynamic optimal bidding strategy (Song et al., 2012) and the **Amazing** (Tang et al., 2012) strategy which proposed an optimal cost reduction for average completion time. Amazing basically does not propose to solve the problem with a constrained deadline. Therefore, it often forces to use on-demand instance when leftover time is equal to the deadline. For a fair comparison, we introduced two different values $(b = 0)$ and $(b = 0.5)$ which represents the percentage of extra time margin when deciding the bidding price with

Amazing. For example, when $b = 0$ the actual execution time is used and the likelihood of switching to on-demand instance is high. In the contrary, when the bid price is estimated with $b = 0.5$, less on-demand instances will be required. We also include naive scheduling algorithms such as bidding at the maximum price, the mean price among al the bidding prices, a randomly selected bidding price and using on-demand instances only, labeled respectively as **Max**, **Mean**, **Random** and **On-demand**. The baseline experiment consists of running jobs that requires $100hrs$ to complete. We vary the deadline D from 100 to 200 by a step of 10. The reason for doing so, is to measure different levels of looseness of the deadline and to capture behavior of the strategies at each point.

4.2 Monetary Cost Analysis

In Fig. 3, we present the average cost of 1000 runs for all compared algorithm. **Static** outperforms **Dynamic** by a small 2% of cost reduction. In all experiments, **Static** performs the best or equal to **Dynamic**. The reason lies on the restart overhead carried out in dynamic bidding strategy. In fact, **Dynamic** can make better decisions on which interval to bid or give up, however, it requires a restart each time a new bid is introduced. Bidding, at **on-demand** price is unaware of the deadline flexibility and therefore the cost remains high when deadline is loosened. In contrast, by adopting the **Mean** price bidding strategy, the cost is proportionally decreased as the deadline is loosened. **Amazing** strategy performs better than **Max**, **Min** and **Random**. However, **Amazing** still costs higher than **Static** and **Dynamic**. **Random** price bidding strategy results in a high cost when deadline is loosened. That's because the **Random** price bidding strategy is oblivious to the execution time and the deadline. In order to minimize the cost our strategy takes into consideration both the execution time the deadline and the captures the price fluctuations. That is also the case of **Dynamic** with a minor difference that a new bidding price is introduced at each time epoch, which leads to restart conditions.

Next, we evaluate the cost saving by the compared algorithms. In Fig. 4, we present the cost saving as compared to the optimal **Static** bidding strategy. Based on our observations, the cost saving as compared to **Max** and **Mean** price bidding strategies are inversely proportional. This is expected because as deadline is tight, **Mean** quickly turns to use more on-demand instances and cost high. At the same time **Max** will use the first T market prices. However, as the deadline is loosened, **Mean** and **Static** both reduce the cost by avoiding high price intervals. Therefore, the cost saving of **static** over **Mean** reduces as deadline is loosened. In contrary, **Max**, invariably to the deadline factor, will still use the first T intervals. As such, the cost saving increased because **Static** is aware of the deadline factor and minimizes the cost accordingly.

As expected using on-demand instance results in higher cost than all other bidding strategies in Spot instance except for tight deadline like when the ratio E/D is 1. In this case, using spot instance will introduce additional cost for restarts and checkpoints. The **Max** bid strategy fails to capture the market price fluctuations . The rational in bidding on the Maximum bidding price is that the user still pays the market price, instead of the bid price. As such, users can achieve high availability while paying less than the **Max**. However, the main reason bidding at the maximum price is not suitable strategy for workload with flexible deadline is that user still pays for higher price rather than postponing for potential cheaper price. There is inherently a dual-option bid when bidding on spot instances as described by ShaoJie Tang et al. (Tang et al., 2012) which consists for each time epoch to decide to bid or not depending on the job urgency and the current price. However, bidding at the maximum price withdraws from this principle. The conclusion we draw from these naive algorithms is that a bidding strategy is needed.

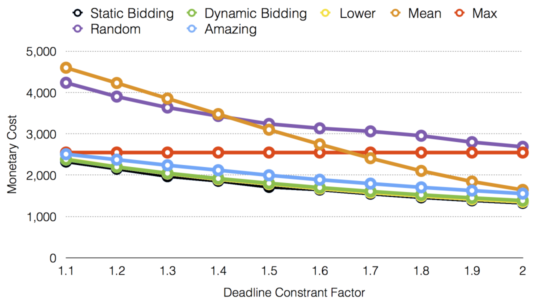

Figure 3: Monetary cost.

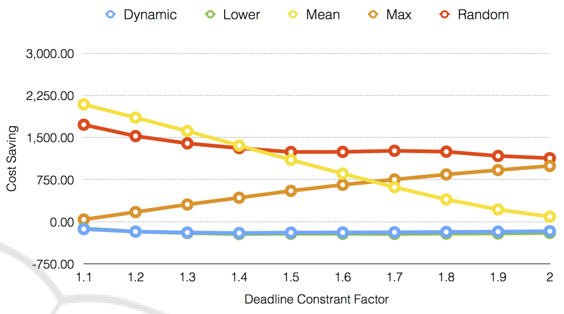

Figure 4: The amount of cost Saving from the optimal static bidding strategy.

Amazing strategy has a higher cost than our strategy because **Amazing** cannot guarantee the deadline. Therefore in most of the cases with tighter deadlines, **Amazing** will switch quickly to on-demand instance. Depends on the buffer setting of **Amazing**. It could perform the best, but also could be worst. Nevertheless, **Amazing** has much larger variance than other compared strategies because it bids on the highest spot price. It shows **Amazing** is a more risky bidding strategy and will rely more often on on-demand instances to complete the job.

Static has similar or even lower cost than **Amazing** and **Dynamic**. It shows that dynamically changing the price is not necessary, and could cause additional computation and running overhead as explained in section 2.1.

5 RELATED WORK

Cloud economies has drawn a tremendous attention over the past decade (Regev and Nisan, 1998; Stokely et al., 2009b; Chaisiri et al., 2011; Genaud and Gossa, 2011; Chard et al., 2015; Oh et al., 2022). Part of this is attributable to the dynamic nature of cloud and the vast pool of underutilized resources in data centers (Stokely et al., 2009a). In auction-based cloud economies such as Spot Instances (SI), availability and cost saving are difficult tradeoff to make (Andrzejak et al., 2010b; Chaisiri et al., 2011; Genaud and Gossa, 2011) since it involves understanding the market fluctuations and the resource model including the availability. Nevertheless, spot instances still provide cheaper resources (Lu et al., 2013). It is therefore important to design a bidding strategy to benefit from this resource model.

6 CONCLUSION

In this paper, we proposed a Static Bidding strategy to efficiently decide the bid price for executing deadline-constraint jobs on Spot Instances. The inputs of our problem are the estimated job execution time and the deadline to complete the job. With these information, we use the historical market data to map the price fluctuation in to random variable based on the Markov chain decision process. Furthermore, to guarantee the completion of the job within the constrained deadline, on-demand instances are used when the deadline is close. We have modeled and included these considerations in the optimal solution. We then pointed out the recursive behavior of solution. Next, we prove that by a means of a Dynamic Programming algorithm, the optimal bidding price can be derived with the checkpoint and restart overhead. Our evaluations conducted with real historical traces from Amazon Spot markets show that Static Bidding strategy can significantly reduce the execution cost on Spot Instance, guarantee the deadline requirement and effectively cope with the checkpoint/restart overheads. As compared to the dynamic bidding strategy, our method is more practical and suitable to the current spot markets. Moreover, we can achieve similar or even better result than the existing strategies.

REFERENCES

Amazon (2004). Amazon console.

Andrzejak, A., Kondo, D., and Anderson, D. (2010a). Exploiting non-dedicated resources for cloud computing. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 341–348.

Andrzejak, A., Kondo, D., and Yi, S. (2010b). Decision model for cloud computing under sla constraints. In *MASCOTS*, pages 257–266.

Chaisiri, S., Kaewpuang, R., Lee, B.-S., and Niyato, D. (2011). Cost minimization for provisioning virtual servers in amazon elastic compute cloud. In *MASCOTS*, pages 85–95.

Chard, R., Chard, K., Bubendorfer, K., Lacinski, L., Madduri, R., and Foster, I. (2015). Cost-aware cloud provisioning. In *IEEE International Conference on e-Science*, pages 136–144.

Chohan, N., Castillo, C., Spreitzer, M., Steinder, M., Tantawi, A., and Krintz, C. (2010). See spot run: Using spot instances for mapreduce workflows. In *HotCloud*, pages 7–7.

EC2, A. (2009). Amazon ec2.

Genaud, S. and Gossa, J. (2011). Cost-wait trade-offs in client-side resource provisioning with elastic clouds. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 1–8.

Jangjaimon, I. and Tzeng, N.-F. (2015). Effective cost reduction for elastic clouds under spot instance pricing through adaptive checkpointing. *IEEE Transactions on Computers*, 64(2):396–409.

Javadi, B., Thulasiramy, R., and Buyya, R. (2011). Statistical modeling of spot instance prices in public cloud environments. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 219–228.

Lu, S., Li, X., Wang, L., Kasim, H., Palit, H., Hung, T., Legara, E., and Lee, G. (2013). A dynamic hybrid resource provisioning approach for running large-scale computational applications on cloud spot and on-demand instances. In *ICPADS*, pages 657–662.

Mazzucco, M. and Dumas, M. (2011). Achieving performance and availability guarantees with spot instances. In *HPCC*, pages 296–303.

Oh, K., Zhang, M., Chandra, A., and Weissman, J. (2022). Network cost-aware geo-distributed data analytics system. *IEEE Transactions on Parallel and Distributed Systems*, 33(6):1407–1420.

Regev, O. and Nisan, N. (1998). The popcorn market&mdash;an online market for computational resources. In *Proceedings of the First International Conference on Information and Computation Economies*, ICE '98, pages 148–157.

Song, Y., Zafer, M., and Lee, K.-W. (2012). Optimal bidding in spot instance market. In *INFOCOM*, pages 190–198.

Stokely, M., Winget, J., Keyes, E., Grimes, C., and Yolken, B. (2009a). Using a market economy to provision compute resources across planet-wide clusters. In *IPDPS*, pages 1–8.

Stokely, M., Winget, J., Keyes, E., Grimes, C., and Yolken, B. (2009b). Using a market economy to provision compute resources across planet-wide clusters. In *IPDPS*, pages 1–8.

Tang, S., Yuan, J., and Li, X. Y. (2012). Towards optimal bidding strategy for amazon ec2 cloud spot instance. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 91–98.

Yi, S., Andrzejak, A., and Kondo, D. (2012a). Monetary cost-aware checkpointing and migration on amazon cloud spot instances. *IEEE Transactions on Services Computing*, 5(4):512–524.

Yi, S., Andrzejak, A., and Kondo, D. (2012b). Monetary cost-aware checkpointing and migration on amazon cloud spot instances. *Services Computing, IEEE Transactions on*, 5(4):512–524.