A Framework for Amazon EC2 Bidding Strategy under SLA Constraints

Shaojie Tang, *Member*, *IEEE*, Jing Yuan, Cheng Wang, and Xiang-Yang Li, *Senior Member*, *IEEE*

Abstract—With the recent introduction of Spot Instances in the Amazon Elastic Compute Cloud (EC2), users can bid for resources and, thus, control the balance of reliability versus monetary costs. Mechanisms and tools that deal with the cost-reliability tradeoffs under this scheme are of great value for users seeking to reduce their costs while maintaining high reliability. In this paper, we propose a set of bidding strategies under several service-level agreement (SLA) constraints. In particular, we aim to minimize the monetary cost and volatility of resource provisioning. Essentially, to derive an optimal bidding strategy, we formulate this problem as a Constrained Markov Decision Process (CMDP). Based on this model, we are able to obtain an optimal randomized bidding strategy through linear programming. Using real Instance price traces and workload models, we compare several adaptive checkpointing schemes in terms of monetary costs and job completion time. We evaluate our model and demonstrate how users should bid optimally on Spot Instances to reach different objectives with desired levels of confidence.

Index Terms—Cloud computing, bidding strategy, EC2

1 Introduction

In December 2009, Amazon released Spot Instances, which is a new way to purchase and consume Amazon EC2 Instances [1], [8]. They allow customers to bid on unused Amazon EC2 capacity and run those Instances for as long as their bid exceeds the current Spot Price.

Service-level agreement (SLA) is an agreement between and service provider and customer, specifying the level of delivered service. In this paper, we assume that the service level is decided by two metrics: *monetary cost* and *execution delay*. Obviously, there is a tradeoff between those two metrics. Therefore, we propose a bidding strategy, called *AMAZING*, that can minimize the expected monetary costs within certain delay bound, which is set by the customers. Based on real price traces, we find that our proposed strategy achieves the lowest monetary cost under various delay requirements.

In our study, we observe that typical state pattern of Spot Instances exists in Amazon EC2. The state pattern information, if utilized in an intelligent manner, can improve the execution efficacy while reducing monetary costs. For instance, the problem of making proper bid decision is formulated as a Constrained Markov Decision Process (CMDP) [7]. After solving the CMDP, *AMAZING* applies optimal bid decision to each Instance hour during the course of the job's computation. Our state context aware design tries to intelligently adapt the bid price using intelligence from detected state patterns.

In this perspective, we have designed an efficient state-pattern-aware bidding strategy *AMAZING*. The novelty of *AMAZING* lies in the intelligence that it learns and adapts from the transition of Instance states, to make the bid decision for each Instance. In our proposed system, an AI agent is trained to extract Spot Price patterns from the dynamically changed Spot Price data. The agent provides Spot Price transition probability matrix (STPM), which contains information about transition probability of Spot Prices for a particular Instance type.

It is critical to control the balance of reliability versus monetary costs. Previous researchers investigate probabilistic model [9] and checkpointing mechanisms [20], [21], [22] to answer the question of how to bid given these constraints. Given the maximum price that users are willing to pay per hour, researchers tend to apply probabilistic model and different checkpointing strategies to meet the requirements. Nevertheless, previous approaches [20], [21], [22] were considered under the fixed bid price model, and only periodically checkpointing schemes were discussed in their study. In this work, we focus on designing an optimal bidding strategy that utilizes both the dynamic pricing model and the state transition intelligence. By formulating the bidding process as a CMDP, we are the first to provide a provably optimal bidding strategy. In particular, we are able to show that by adopting our bidding strategy, the expected execution delay can be bounded and at the same time, the monetary costs are minimized. Most importantly, our strategy is easy to compute and, thus, very effective in real implementation.

2 System Model of Spot Instance

In this section, we describe the work model adopted by Spot Instance.

- *S. Tang is with Temple University, Room 312 Wachman Hall, 1805 N Broad Street, Philadelphia, PA 19122. E-mail: stang7@hawk.iit.edu.*
- *J. Yuan is with the Department of Computer Science, University of Illinois, 7901 Henry Ave Apt D401, Philadelphia, PA 19128. E-mail: jyuan5@uic.edu.*
- *C. Wang is with the Department of Computer Science and Engineering, Tongji University, Room 1810, Building 13, Lane 630, Quyang Road, Shanghai, 200092, China. E-mail: 3chengwang@gmail.com.*
- *X.-Y. Li is with the Department of Computer Science, Illinois Institute of Technology, Room 229C Stuart Building, 10 West 31st Street, Chicago, IL 60616. E-mail: xli@cs.iit.edu.*

Manuscript received 30 May 2012; revised 19 Dec. 2012; accepted 22 Dec. 2012; published online 11 Jan. 2013.
Recommended for acceptance by L.E. Li.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2012-05-0513.
Digital Object Identifier no. 10.1109/TPDS.2013.15.

TABLE 1
Instance Types (OS: Linux Class: Hi-CPU= HIGH-CPU, Std= STANDARD, Hi-Mem= HIGH-MEMORY)

Symbol	Class	API Name	Mem. (GB)	Total Units	Num. Cores	Units/ Core
A	std	m1.small	1.7	1	1	1
B	std	m1.large	7.5	4	2	2
C	std	m1.xlarge	15	8	4	2
D	hi-cpu	c1.medium	5	2	2.5	2.5
E	hi-cpu	c1.xlarge	7	20	8	2.5
F	hi-mem	m2.xlarge	17.1	6.5	2	3.25
G	hi-mem	m2.2xlarge	34.2	13	4	3.25
H	hi-mem	m2.4xlarge	68.4	26	8	3.25

2.1 Characteristics of the Spot Instances

Spot Instances enable users to bid for unused Amazon EC2 capacity. Amazon provides 64 types of Spot Instances that differ by computing/memory, OS type and geographical location [17]. We summarized in Table 1 different types of Spot Instances that possess varied processing capacity. As expected, to obtain more powerful Spot Instances, users need to bid higher prices. Fig. 1 illustrates partial price history for some Spot Instance types. The data we use to plot this figure are collected from a website [5] that is used to display interactive charts of Amazon Spot Instance prices. As shown in the figure, Spot Instances are charged the Spot Price set by Amazon EC2, which fluctuates periodically depending on the supply of and demand for Spot Instance capacity. In specific, Amazon EC2 will change the Spot Price periodically as new requests are received and as available Spot capacity changes (e.g., due to Instance terminations). While the Spot Price may change anytime, in general the Spot Price will change once per hour, which is named as *Instance hour* in this work, and in many cases less frequently. To use Spot Instances, the users place a Spot Instance request, specifying the Instance type, the region desired, the number of Spot Instances they want to run, and the maximum price they are willing to pay per hour. To determine how that maximum price compares to past Spot Prices, the Spot Price history is available via the Amazon EC2 API and the AWS Management Console. If the maximum price bid of the user is higher than the current Spot Price, his/her request is fulfilled and his/her Instances will run until either you choose to terminate them or the Spot Price increases above his/her maximum price (whichever is sooner).

Fig. 2 illustrates how Amazon EC2 charges *per-hour* price for using a Spot Instance. It is important to note the following characteristics of Amazon EC2's Spot Instances which are also highlighted in [8].

√ When a user's maximum price bid is higher than the current Spot Price, he/she is granted the access to the requested resource. Otherwise, Amazon terminates the service when a user's bid is under the current Spot Price. We call this a failure (or out-of-bid). To ensure the reliability of task execution, users are suggested to save their intermediate results periodically by means of checkpointing [8], [9].

√ Spot Instances can be terminated when the user no longer requests them. If the user terminates his/her Instance, he/she will pay for any partial hour. However, if the Spot Price goes above the user's maximum price and the Instance is terminated by Amazon EC2, he/she will not be charged for any partial hour of usage.

√ The price of a partial hour is considered the same as a full hour. Amazon charges each hour by the current Spot Price. The price of Amazon's storage service is negligible (at most 0.15 USD per GB-month) which is much lower than the price of computation [2].

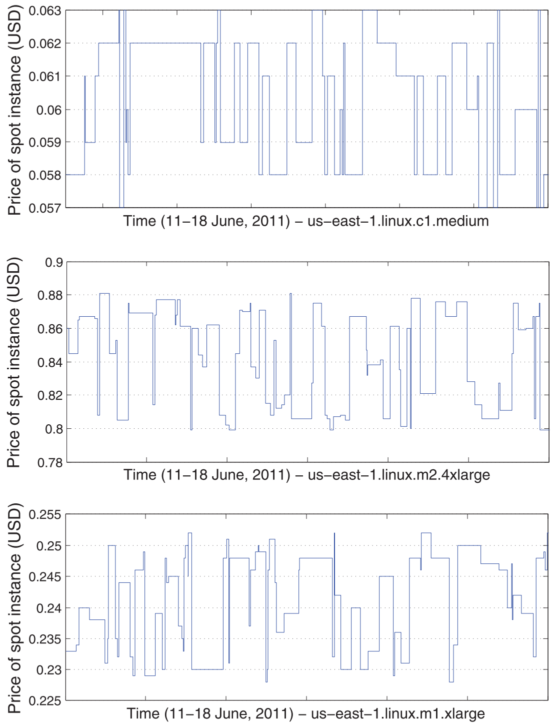


Fig. 1. Spot Price fluctuations of *us-east-1.linux* Instance types.

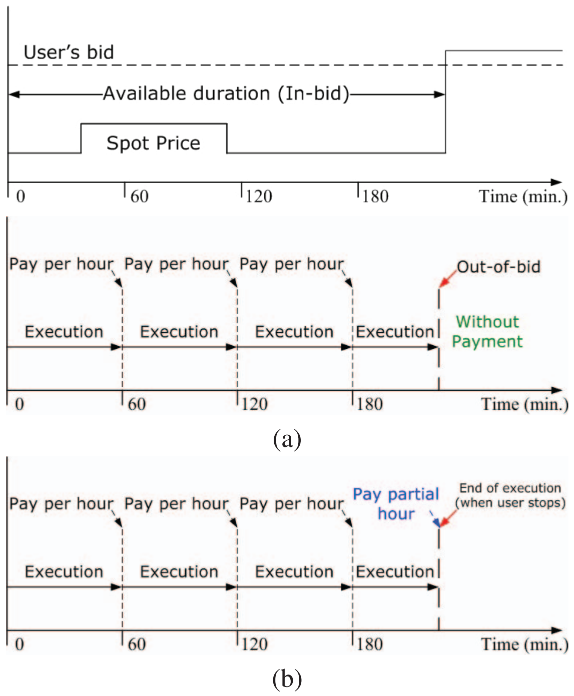


Fig. 2. Examples of pricing for Amazon EC2 Spot Instances: (a) When a user's Spot Instance is out-of-bid; (b) when a user stops using Spot Instance.

Shed light from above characteristics of Spot Instances, previous researchers proposed a potential exploitation method to reduce the cost of the last partial hour of work [9]. In this work, we utilize the same strategy to potentially prevent a payment in the last partial hour. In this scenario, a user waits after finished computation almost to the next hour-boundary for a possible termination due to an out-of-bid situation.

2.2 Definitions

We assume a user is submitting a parallel, compute-intensive job that is divisible. Divisible workloads such as video encoding, advanced graphics, and virtual reality are an important class of application prevalent in high-performance parallel computing. This is a common type of application that could be submitted on EC2 and amenable to failure-prone Spot Instances [9].

Suppose that the job consists of a total amount of work W to be executed. W can be divided into N_p many Instances of the same type in parallel, which gives the workload per Instance as $W_p = W/N_p$. Note that N_p can be usually varied up to a certain limit given by the number N_{max} of "atomic" tasks in the job, so $N_p < N_{max}$. We measure W and W_p in hours of computation needed on a single EC2 Instance with processing capacity of 2.5 EC2 Compute Units, i.e., a single core of the High-CPU medium Instance type. We refer to the time needed for processing W_p on a particular Instance type I_t the task length $T = T(I_t)$. Suppose that the processing capacity of I_t is C_t, we have the perfect relationship $T(I_t) = W_p/C_t$. It is worth to note that T is the net computation time excluding any overheads due to out-of-bid, checkpointing, and recovery. This is different than the actual time needed to process W_p, which is called *execution time*. Letting t_d denote the deadline of a job to finish (for a fixed Instance type), and T_e the execution time, our objective is to minimize the total monetary cost while satisfying $T \le E[T_e] \le t_d$. Further, we denote by t_c the time for taking a checkpoint, and t_r the time for a restart.

To emulate real applications, we base the input workload on that collected in real Grids and Desktop Grids. The majority of workloads in traditional Grids consist of pleasantly parallel and independent batches of tasks. The mean job deadline t_d from the Grid workloads archive [13] is 1,074 minutes (17.9 hours). The mean task length T is 164 minutes (2.7 hours) on a 2.5-GHz core.

In this work, we denote the maximum Spot Price of a particular Instance type by p_{max} (exclude the burst noises). Further, for a specific Instance type, the Spot Price for the ith hour is denoted by p_i. We define the *result* of each hour during the course of the job's computation as *in-bid* or *out-of-bid*. As discussed in Section 2.1, if the user's bid exceeds p_i, the *result* of the ith hour becomes *in-bid*. Conversely, the *result* of the ith hour becomes *out-of-bid* if the user's bid is below p_i. We use $x^{\alpha,\beta}_{p_i}$ to denote an *integrated state* (or simply *state*) of the ith hour. By α and β, we denote the *result* of the $(i-1)$th hour and the ith hour, respectively. Note that $\alpha, \beta \in \{i, o\}$, where i is short for *in-bid* and o is short for *out-of-bid*. For instance, $x^{o,i}_{p_i}$ means that after *out-of-bid* in the $(i-1)$th hour, it becomes *in-bid* in the ith hour, and the Spot Price is p_i. Integrated the result of the previous hour into the state of each hour, $x^{\alpha,\beta}_{p_i}$ provides a compact means to define the *execution progress* in Section 2.4.

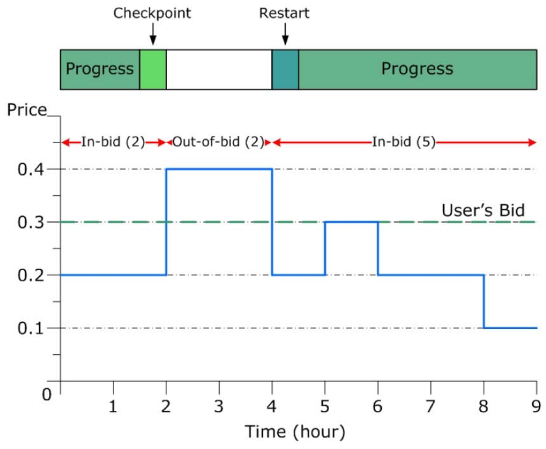


Fig. 3. Probability density function $f(t, p)$ of failure occurrences, where t is the time since the last checkpoint at bid price p.

2.3 Execution Scenario

Fig. 3 illustrates an exemplary execution scenario. A user submits a job with a total amount of work W of 12 unit-hours with $N_p = 2$, which translates to a $W_p = 6$ unit-hours and the task time (per Instance) T of 6 hours (assuming EC2's small Instance server). User's bid price u_b is 0.30 USD, and during the course of the job's computation, the job encounters an *out-of-bid* situation (i.e., failure) between time 2 and 4. The total availability time was 7 hours, from which the job has $(1.5 + 4.5) = 6$ hours of real execution progress (i.e., useful computation), and uses 0.5 hour for checkpointing and 0.5 hour for restart. Obviously, these overheads are unrealistic, but defined here for simplicity of the example. The execution time T_e needed until finishing was 9 hours. During the job's active execution, the Spot Price fluctuates. There are 1 hour at 0.10 USD per time unit, 5 hours at 0.20 USD per time unit, and 1 hour at 0.30 USD per time unit, giving a total cost of 1.4 USD. Thus, the expected price is $1.4/7 = 0.2$ USD per hour. The expected progress $E[f] = T/T_e = 6/9 \doteq 0.67$.

2.4 Validation of the Dual-Option Strategy

To explore an optimal bidding strategy, we first need to investigate how to make a reasonable bid decision for each hour. Theorem 1 implies that we can obtain an optimal bidding sequence by making a choice from exactly two options for each hour: bidding the maximum Spot Price or bidding zero dollars (give up).

Theorem 1. *Given the Spot Price history of a particular type of Spot Instance, the budgetary constraint and deadline requirement t_d, an optimal bidding sequence that minimizes monetary cost while meeting the deadline can be obtained by making a choice from exactly two options for each hour: bidding the maximum Spot Price or bidding zero dollars (give up).*

Proof. Detailed proof can be found in Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.15. □

In this work, our Spot Price context aware strategy tries to intelligently adapt the bid price using intelligence from detected pricing patterns. During each Instance hour, our dual-option *AMAZING* algorithm precomputes the desired probability of each bid option for the next Instance hour (bid or give up) based on the current Instance state information. Let $\mathcal{A}$ denote the set of the options that we consider each time. Implied by Theorem 1, A is composed of only two elements, i.e., $\mathcal{A} = \{B, G\}$. Here, B is short for *bid the maximum Spot Price* and G is short for *give up*. We denote the bid decision made in the ith hour by a_i where $a_i \in \mathcal{A}$.

Since we always make our bid decision ahead of time, we can make a checkpoint in the current hour whenever we decide to give up (i.e., bid 0 USD) in the next hour. We define the *execution progress* for the ith Instance hour, i.e., $f(x_i, a_i)$ in following equation (x_i denotes the integrated state of the ith hour):

$$f(x_i, a_i) = \begin{cases} 1 - t_c, & \text{if } x_i = x_{p_i}^{i,i}, a_i = G \\ 1 - t_r, & \text{if } x_i = x_{p_i}^{o,i}, a_i = B \\ 1 - t_c - t_r, & \text{if } x_i = x_{p_i}^{o,i}, a_i = G \\ 1, & \text{if } x_i = x_{p_i}^{i,i}, a_i = B \\ 0, & \text{otherwise.} \end{cases}$$

3 Bidding Strategies

3.1 Always Bidding On-Demand Price

The first strategy, namely always bidding on-demand price (AO), is very simple and effective. As its name implies, we always bid at the maximum price.

The motivation of this strategy can be summarized as follows: If current Spot Price is lower than the bid price, we will be charged the current price regardless of our bid. Notice that we are making an assumption that the Spot Price will never go above the on-demand price. This is simply because no one will ever bid equal to or more than the on-demand price because then they can simply provision an on-demand Instance.

Lemma 2. *For any job Instance, AO strategy guarantees 1) minimum completion time and 2) at most 10 percent more than the minimum cost.*

Proof. Detailed proof can be found in Appendix, available in the online supplemental material. □

3.2 Adaptively Setting Backup Point—Valley Bidding

We next propose a backup point-based method to reduce the cost while minimizing the job completion time. Here, checkpoints are taken periodically at hour boundaries. It is the most intuitive one for the Spot Instances because an hour is the lowest granularity of Spot Instance pricing. It also provides a guarantee of paying for the actual progress of computation. A variation of this policy is the fine-grained checkpointing, which evaluates whether to take a checkpoint periodically every 10 or 30 minutes.

Valley bidding, as its name says, is a bidding strategy for which a user bid a maximum price only when the Spot Price decreases. In the best case, the Spot Price drops to the bottom of the *valley*, the user will be charged at the lowest price for the Instance hours she obtained. Conversely, once the Spot Price increases, the user will give up bidding. During the Instance hour that right before the giving up, a checkpoint will be taken to preserve the intermediate computation results. In this way, valley bidding facilitates the decision making by means of avoiding potential high prices.

3.3 CMDP-Based Bidding Strategy

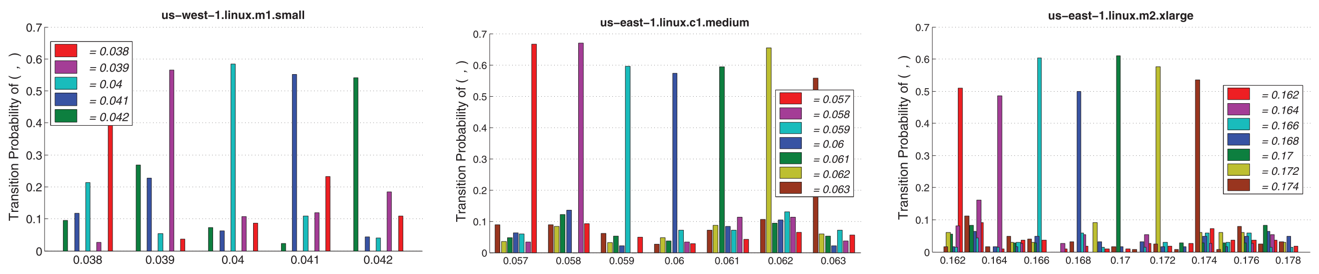
The goal of *AMAZING* is first reinstated with formulation. Recall that we defined t_d as the execution time constraint (i.e., deadline), which is the maximum expected clock time allowed. We study the following constrained optimization problem: *considering a long state evolving process, given an execution time constraint t_d and STPM, what is the optimal bid strategy μ, such that the expected monetary cost to finish the entire task $E[M]$ is minimized, and the expected execution time is maintained under the constraint, i.e., $E[T_e] \leq t_d$?*

It is a critical challenge to control the balance of reliability versus monetary costs. Previous researchers investigate probabilistic model and checkpointing mechanisms to answer the question of how to bid given these constraints. Given the maximum price that users are willing to pay per hour, researchers tend to apply probabilistic model [9] and different checkpointing strategies [20] to meet the requirements. Nevertheless, previous approaches were considered merely under the fixed-bid price model, and only empirical results were given in their study. In this work, we try to design an optimal bidding strategy that utilizes both the dynamic pricing model and the state transition intelligence. For each Instance hour, a bid decision is made based on *STPM* for the next Instance hour. In this way, the appropriate time for checkpointing can be known ahead of time. By preserving the intermediate computing results in time, we are able to improve the execution efficacy.

The program running *AMAZING* just needs to be informed of the integrated state of each Instance hour. Based on the collected intelligence about state transitions in *STPM*, a back-end system runs a linear programming (LP) routine to select the bid option (make a desirable decision for the next Instance hour). After learning phase, *AMAZING* reduces the total monetary cost for running a submitted task in a long period.

The *STPM* for a particular Spot Instance type can be learned from Spot Price history released on Amazon EC2 website. For a particular Instance type, Spot Prices fluctuate within a small range through a relatively long period. With E different Spot Prices for a particular Instance type, we can build an $E \times E$ matrix *STPM* to represent the transition probability between each Spot Price pair s_i and s_j (for $1 \leq i, j \leq E$). Fig. 4 shows the transition probability of partial Spot Price pairs (s_i, s_j). The results shown in the figure are computed from the trace price history for various Instance types in June 2011. For instance, as shown in Fig. 4b, the difference between the highest and the lowest Spot Price for *us-east-1.linux.c1.-medium* Instance type is within 0.006 USD per hour through last month (exclude burst noises). In this example, we have $0.057 \leq s_i, s_j \leq 0.063$ (for $1 \leq i, j \leq 7$). For ease of illustration, we use (σ, τ) in the figure to denote the price pairs. Given a particular value of σ in an Instance hour, we compute the probability that the Spot Price transit to τ in the following Instance hour.

Our approach builds the matrix based on collected history trace data from Amazon. In this way, we can



(a) (b) (c)

Fig. 4. Transition probability of (p_i, p_j) which represents the probability that Spot Price p_i is immediately followed by Spot Price p_j for different types of Spot Instances.

estimate the probability of transition between two Spot Prices s_i and s_j based on the relative frequency. This probability of transition would be represented by an entry in the matrix. We denote each entry in *STPM* by $\delta(s_i, s_j)$, which is estimated using the formula in

$$\delta(s_i, s_j) = \frac{|s_i \text{ followed by } s_j \text{ in adjacent hours}|}{|\text{Spot Price equals to } s_i|}. \quad (1)$$

Clearly, an entry $\delta(s_i, s_j)$ of a higher value indicates a higher transition probability from Spot Price s_i to s_j. Also note that $\delta(s_i, s_j) \neq \delta(s_j, s_i)$ may hold for most Spot Price pairs. We further observe that $\delta(s_i, s_i) \geq 50\%$ tends to be satisfied in the *STPM* of different Instance types. In our proposed approach, an AI agent is trained to extract Spot Price patterns from the dynamically changed Spot Price data.

In each Instance hour, based on the state of current Instance, the algorithm computes the desired probability of different bid options for the next Instance hour. The detailed selection of a proper bid option is explained in the next section based on the decision making strategy μ. Then, corresponding checkpoint should be taken if we decide to give up in the next Instance hour. It is worth noting that when our algorithm computes the desired probability of each bid option, it has considered all the overhead (e.g., checkpoint) that would be generated. By making a checkpoint during the end of current Instance hour, we are able to preclude the loss of intermediate results.

In Algorithm 1, μ specifies the desired probability of each bid option for the next Instance hour. We model the computation of optimal bid strategy problem as a CMDP. By solving the corresponding LP in polynomial time [7], we obtain an optimal strategy μ for each state. For each state, the selection of different bid option is randomized under fixed distribution.

Algorithm 1. Pseudocode for *AMAZING* Policy.
Input: Optimal bidding strategy μ (computed from Algorithm 2) and current state information
Output: The bidding decision for the next hour

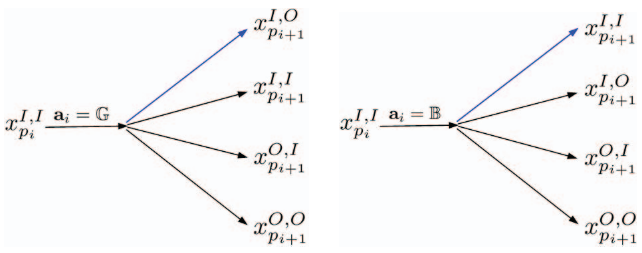
```
1: Randomly choose a bid decision based on bidding
   strategy $\mu$ calculated from Algorithm 2 for the next
   Instance hour;
2: if the bid decision is to give up in the next Instance
   hour then
3:   make a checkpoint at the end of current Instance hour;
4: Perform bidding decision correspondingly in the next
   hour.
```

3.3.1 Constrained Markov Decision Process

Markov decision processes (MDP), also known as controlled Markov chains, constitute a basic framework for dynamically controlling systems that evolve in a stochastic way. We focus on discrete time models: we observe the system at stages $t = 0, 1, 2, \ldots, n$, where n is called horizon, and may be either finite or infinite. A controller has an influence on both the costs and the evolution of the system, by choosing at each time unit some parameters, called actions. As is often the case in control theory, we assume that the behavior of the system at each time unit is described by a *state*, as well as the *action*. The system moves sequentially between different states in a random way, current state fully determines the probability to move to any given state in the next time period unit. In a standard MDP, current action may also affect the transition probability for the next time unit, but this is not the case in this paper where the transition matrix does not depend on current decision making algorithm. MDP is, thus, a generalization of (noncontrolled) Markov chains, and many useful properties of Markov chains carry over to controlled Markov chains. The model that we consider in this paper is special in the sense that more than one objective cost exists, and the controller minimizes one of the objectives subject to constraint the other.

To make the above framework fit our problem setting, we define a tuple $\{O, X, \mathcal{A}, P, M, D\}$, where: 1) $O = \{t \mid t = 1, 2, \ldots\}$ denotes the set of decision epochs. Decisions are made at the beginning of each Instance hour. 2) $X = \{x_1, x_2, \ldots, x_n\}$ is a countable state space. Although we limit the study in this work to discrete Instance state transitions, the continuous case can also be handled by dividing it to discrete space. Recall that we gave a preliminary definition of x_i in Section 2.2. We have $x_i = x_{p_i}^{\alpha,\beta}$ for $x_i \in X$. 3) $\mathcal{A}$ is a metric set of actions. We defined $\mathcal{A} = \{B, G\}$ as the compact option set under consideration in Section 2.4. Each element $a_i \in \mathcal{A}$ defines a bid option, i.e., either *bid* or *give up* for the next Instance hour. Note that a_i, the bid decision made in the ith Instance hour, will directly affect the state of the $(i+1)$th Instance hour.

Theoretically, each Instance hour has $4 \cdot E$ possible states. As defined previously in Section 3.3, E is the number of different Spot Prices that presented in the price history of a particular Instance type. Since $\alpha, \beta \in \{i, o\}$, the number of different combinations of α, β is 4. Thus, we have $4 \cdot E$ possible states for each Instance hour. To reduce the searching space, we leverage underlying transition matrix to facilitate our study.



(a) (b)

Fig. 5. Different bid decisions made in the current Instance hour will lead to different possible state spaces of the next Instance hour.

We use the previous example in Fig. 4a for illustration. In this example, a possible bid option a_i could be *give up*, i.e., $a_i = G$. Since $x_i = x^{i,i}_{p_i}$ and a_i directly determines the state of the next Instance hour to be $x_{i+1} = x^{i,o}_{p_{i+1}}$, the size of searching space is reduced to E. As illustrated in Fig. 5, different bid decisions made in the ith Instance hour will lead to different possible state space of the $(i+1)$th Instance hour.

Now, 4) Let $\rho(x_i, a_i)$ denote the *occupation measure* of state x_i and bid option a_i, i.e., the probability that such state-option pair ever exists in the decision process. $E[f]$ denotes the *expected execution progress* in an Instance hour, which is expressed as in (2). Notice that the *occupation measure* $\rho()$ is decided by corresponding decision making strategy. 5) P represents the state transition probability matrix (different from STPM). Define $P_{i,j}(a_i)$ as the probability of moving from Instance state x_i to x_j, when bid option a_i is taken. Here, x_j denotes any *possible* state of the $(i+1)$th Instance hour. If x_j is a *valid* state given x_i and a_i, then $P_{i,j}(a_i) > 0$. Otherwise, $P_{i,j}(a_i) = 0$.

Take Fig. 5 as an example. As illustrated in Fig. 5a, *possible* states of the $(i+1)$th Instance hour include all $4 \cdot E$ states on the right-hand side of the arrows. Among them, only E states of the $(i+1)$th Instance hour is *valid*. More specifically, only $x_{i+1} = x^{i,o}_{p_{i+1}}$ can be reached given $x_i = x^{i,i}_{p_i}$ and $a_i = G$. Similarly, given $x_i = x^{i,i}_{p_i}$ and $a_i = B$, we have another set of E *valid* states for the $(i+1)$th Instance hour. More specifically, only $x_{i+1} = x^{i,i}_{p_{i+1}}$ can be reached in this scenario. In Fig. 5, all *valid* states of the $(i+1)$th Instance hour under different a_i are labeled on the right-hand side of the blue arrows.

Now, 6) M is the immediate cost. In this paper, we define $E[M]$ as the expected total monetary cost during the course of the job's computation, which can be computed as in (2). 7) D is the maximum clock time allowed to complete the job, i.e., deadline. In this paper, we have $D = t_d$. Recall that $f(x_i, a_i)$ denotes the *execution progress* for the ith Instance hour. Please refer to Section 2.4 for detailed calculation of f under different Instance states. Therefore, we have the expected execution progress for each Instance hour $E[f] \geq \frac{1}{t_d}$:

$$\begin{aligned} E[f] &= \sum_{x_i \in X} \sum_{a_i \in A} \rho(x_i, a_i) \cdot f(x_i, a_i), \\ E[M] &= \sum_{x_i \in X} \sum_{a_i \in A} \rho(x_i, a_i) \cdot p_i \cdot t_d. \end{aligned} \tag{2}$$

3.3.2 *Optimal Bidding Strategy* μ

To compute the optimal strategy of the CMDP with expected monetary cost criteria, we can formulate it as a linear programming. After solving the corresponding LP, we can obtain the optimal strategy through normalization ([7, Chapter 4]). We next write the bid optimization problem defined above as the following LP. The constraints (1) and (3) ensure that $\rho(x_i, a_i)$ is a feasible probability measure. The deadline requirement can be respected under constraint (2) by setting the expected execution time less than $\frac{1}{t_d}$. In inequality (4), $\delta_{x_j}(x_i)$ is the delta function of x_i concentrated on x_j, which is defined in following equation:

$$\delta_{x_j}(x_i) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise.} \end{cases}$$

This constraint describes that the outgoing rate and incoming rate for a state must be the same. At the same time, it emphasizes the property for ergodic processes. After solving the linear programming, we get an optimal occupation measure $\rho()$ in terms of monetary cost minimization for each state/bid-option pair. However, since $\sum_{a_i \in \mathcal{A}} \rho(x_i, a_i) \leq 1$, we cannot directly use $\rho(x_i, a_i)$ as the probability of taking action a_i at state x_i.

Problem: *LP-Minimizing Expected Monetary Cost*
Objective: *Minimize* $E[M]$
subject to:

$$\begin{cases} (1)\ \ \rho(x_i, a_i) \geq 0, \ \ \forall x_i, \ \forall a_i \\ (2)\ \ E[f] \cdot t_d \geq 1 \\ (3)\ \ \sum_{x_i \in X} \sum_{a_i \in \mathscr{A}} \rho(x_i, a_i) = 1 \\ (4)\ \ \forall x_j \in X \\ \quad\ \sum_{x_i \in X} \sum_{a_i \in \mathscr{A}} \rho(x_i, a_i)(\delta_{x_j}(x_i) - P_{i,j}(a_i)) = 0 \end{cases}$$

Theorem 3. *The optimal bid strategy μ can be determined from $\rho(x_i, a_i)$ as follows:*

$$\mu(a_i \mid x_i) = \frac{\rho(x_i, a_i)}{\sum_{a_i \in \mathcal{A}} \rho(x_i, a_i)}.$$

Here, μ describes the probability that taking bid option a_i at state x_i.

It is easy to verify that $\sum_{a_i \in \mathcal{A}} \mu(a_i \mid x_i) = 1$. Please refer to [7, Theorem 4.3] for the proof of the correctness of Theorem 3. For any number of input state, Algorithm 2 can return an optimal strategy μ in polynomial time. As input of Algorithm 1, $\mu(a_i \mid x_i)$ for all i will be performed in the next Instance hour.

Algorithm 2. Computation of Optimal Bidding Strategy μ.
Input: Execution deadline t_d, transition matrix P
Output: Optimal bidding strategy μ.

```
1: Solve corresponding CMDP linear programming to get
   the occupation measure $\rho(x_i, a_i)$, $\forall x_i \in X, \forall a_i \in \mathcal{A}$;
2: Calculate optimal bidding strategy $\mu$ from $\rho(x_i, a_i)$ as
```

$$\mu(a_i \mid x_i) = \frac{\rho(x_i, a_i)}{\sum_{a_i \in \mathcal{A}} \rho(x_i, a_i)}$$

4 Performance Evaluation

In our study, we considered prices of Instance types that run under Linux/UNIX OS and are deployed in the zone us-east-1. Table 1 shows the symbols, class (High-CPU, Standard, High-Memory), API names, RAM memory (in GB), total processing capacity in EC2 Compute Units, number of cores, and processing capacity per core. Interested readers please refer to [1] for details. We simulated the bid optimization algorithms based on the real price traces in terms of the task completion time, total price, and the time overhead of checkpointing and restart.

4.1 Evaluation Settings

Table 2 shows our simulation setup in detail. The data set we used in our experiments is collected from a website [5] that interactively display charts of Amazon Spot Instance prices. The time to analyze price history is set to be 3 seconds. And default task restart time is set to be 10 minutes. We collect price history of Instance type D for 52 days to train the STPM. For testing, we collect price history of the same type of Instance for 29 days to evaluate the performance of the proposed bidding model. We assume that the checkpointing cost of running programs is known. We used the constant value for the t_c but using a variable checkpointing cost is also possible in our system model. The minimum price granularity for bidding option is set to be 0.001 USD. If not stated otherwise, we use the Instance type D with task length T of 164 minutes (2.7 hours) in our experiments. Furthermore, our models assume that a job is executed on a single Instance only, as running several Instances of same type in parallel yields the identical time and proportional cost behavior.

4.2 Impact of Constraint Parameters

In this part, we study how the constraint factors such as budgetary constraint affect the distributions of the execution time and monetary cost per Instance. We also investigate the overhead of the checkpoint/restart during the course of the job's computation.

4.2.1 *Execution Time and Monetary Cost*

Fig. 6a shows execution time for various values of budgetary constraints and desired confidence in meeting the job's deadline c_T. Instead of assuming a fixed deadline, we study here the execution time which can be achieved with confidence c_T. As shown in Fig. 6a, low budgets in conjunction with high values of confidence lead to extremely long execution times, which can be up to factor 15 compared to the task length T. For sufficiently high budget ($\geq$0.18 USD), the execution time drops to half of the peak value. Only in the top range of the budgetary constraints, the execution time is on the order of T.

Fig. 6b shows the monetary cost under varied budgetary constraints and desired confidence in meeting the budget c_M. Differently from the execution time, monetary cost increases only slightly with the budget constraint, and is relatively indifferent to the confidence. We explain this by the fact that a long execution time comes primarily from out-of-bid (give up) time. In this scenario, the user is not charged. Even during an execution time of 35 hours there might be only small in-bid time on the order of execution time T that is charged. In conclusion, a user does not save much by bidding low (within 10 percent) but risks very high execution times.

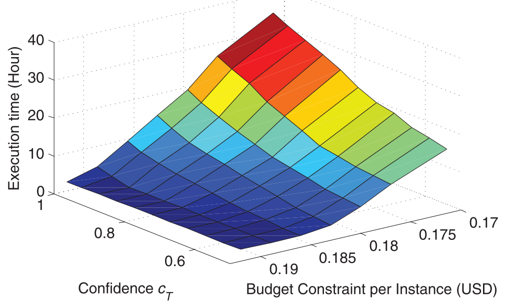
We also find that a slight change of the budgetary confidence c_M has significant impact on the execution time. If the user assumes 0.01 USD more for the budget, she will benefit from a significant reduction of execution time at the same confidence value.

TABLE 2
Values of Parameters Used in This Paper

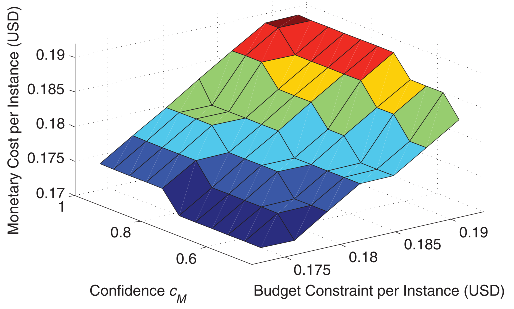
Parameter	Value
Time to take a checkpoint (t_c)	5 mins
Time to analyze price history for obtaining μ	3 secs
Time to restart a task (t_r)	10 mins
Starting date of training traces	March 15th, 2011
Ending date of training traces	May 7th, 2011
Starting date of testing traces	May 21st, 2011
Ending date of testing traces	June 18th, 2011
Testing traces for calculating pdf	10,080 mins
Minimum price granularity	0.001 USD

4.2.2 *Overhead of Checkpoint/Restart*

Fig. 7a shows the overhead of checkpoint/restart during the task execution, where both the budgetary constraint and the confidence of availability (in-bid) time c_I. We denote availability time by AT. We study the time overhead due to checkpointing and restart. We observe that our approach outperforms previous work in terms of low execution time and overhead while meeting the budgetary constraint. This will be discussed in detail later in Section 4.3. Clearly, low

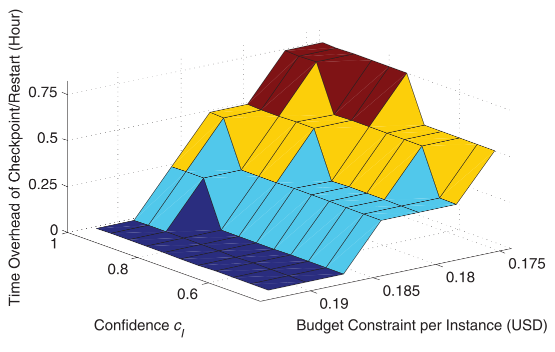


(a)



(b)

Fig. 6. Impact of budget constraint and desired confidence in meeting the job's deadline c_T.



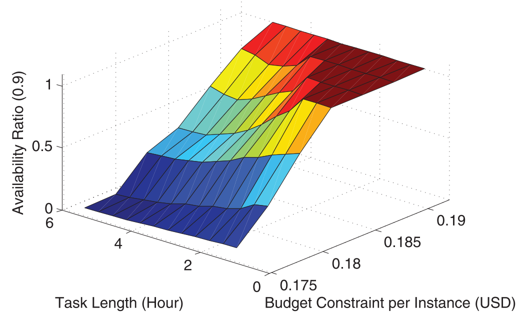


Fig. 7. (a) Overhead of the checkpoint/restart for various budget constraints; (b) Availability ratio depending on the budget constraint and task length T.

budgets lead to more frequent out-of-bid situations, which increases the checkpointing overhead.

4.2.3 *Impact of Task Length*

Fig. 7b illustrates how the distributions of the availability ratio (AR) depends on the task length T. Availability ratio is another importance feature utilized in cloud computing to evaluate the efficacy of bidding algorithms. We first give a T-*free* definition of AR: the ratio of the total time in-bid to execution time, i.e., $AR = AT/T_e$. Fig. 7b shows $AR(0.9)$, i.e., value $v \geq$ (90 percent of values assumed by AR) as a function of the budgetary constraint and T. Here, the impact of T is strongly visible, especially for low budgets. As a consequence, distribution of AR depend on T, and cannot be stored only as functions of T-*free* factors.

4.3 Evaluation of Execution Time

In this section, we study performance of *AMAZING* in terms of the execution time compared to previous work [9], [20]. In previous work, bid price for each Instance hour is fixed and various checkpointing strategies are proposed to balance the reliability and monetary cost during the course of job's computation. The *HOUR* strategy proposed in [20] is the hourly strategy, where a checkpoint is taken at a boundary of each paid hour (measured from the start of current availability interval). They evaluate the performance of this hourly checkpointing strategy *HOUR* with the *OPT* strategy (an unrealistic base-case) for various values of bid price and desired confidence. We conduct extensive experiments to compare our bidding strategy *AMAZING* with the *HOUR* strategy, where both the bid price (*HOUR*) and task length T are varied.

Fig. 8 shows the cumulative distribution function (CDF) of the execution time T_e according to different bidding strategies and task lengths T. The budget constraint for our proposed approach is set to be 0.18 USD per Instance. We used the same settings of the remaining parameters, such as checkpointing cost, rollback cost (restart cost), and processing capacity of the Instance type as in [20], [9]. Obviously, under the same task length T, the *AMAZING* strategy has much lower time overhead. We explain this by the fact that our bid decision for each Instance hour is computed from the Instance state transition probability matrix (*STPM*). *AMAZING* exploits the predicted Spot Price transition to solve the bid optimization problem; thus, it is more likely to help customers skip unnecessary checkpoints/restart when the Spot Price is relatively smooth. Consequently, more clock time is utilized to perform effective computation, and a higher utilization ratio (*UR*) is achieved, where $UR = T/T_e$. On the other hand, the *HOUR* strategy asks customers to take a checkpoint at a boundary of each paid hour without *looking ahead* even if with high probability the Spot Price could be even lower in the next Instance hour. As a result, more than 20 percent of the availability time is taken for hourly checkpointing/recovery, less time is utilized for real program processing. In summary, achieving a higher UR during the course of task computation is the main reason that the proposed *AMAZING* strategy outperforms the *HOUR* strategy. We also observed that the advantage of *AMAZING* is even more obvious when the task length T increases. As shown in Fig. 8, the *HOUR* strategy with various bid prices rarely meets the confidence $c_T = 90\%$ under the budget constraint of 0.18 USD. In contrast,

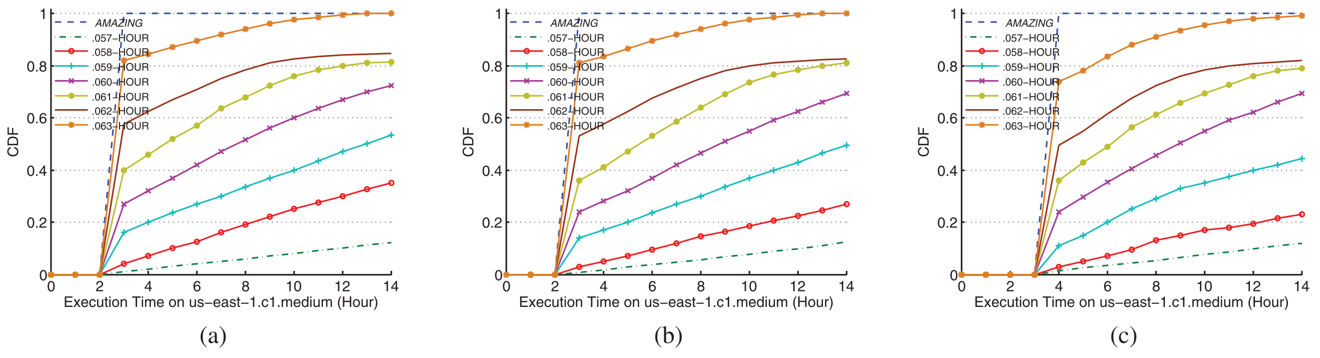


Fig. 8. CDF of execution time for different task length T on Instance type D, where (a) $T = 144$ mins, (b) $T = 164$ mins, (c) $T = 184$ mins.

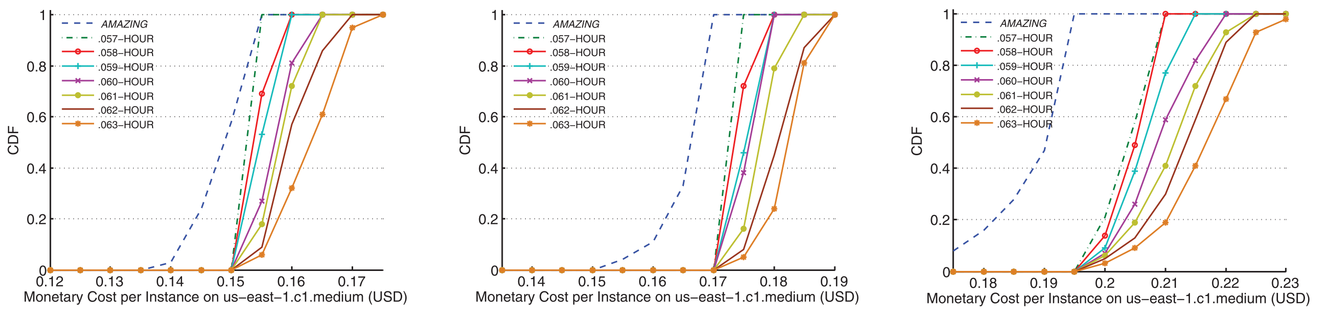


Fig. 9. CDF of monetary cost for different task length T on Instance type D, where (a) $T = 144$ mins, (b) $T = 164$ mins, (c) $T = 184$ mins.

AMAZING reaches 100 percent confidence of meeting the deadline requirement at less than 4 hours.

4.4 Evaluation of Monetary Cost

In this section, we study performance of *AMAZING* in terms of the monetary cost compared to previous work [9], [20]. We used the same settings of parameter as in previous section. In Fig. 9, we illustrated the CDF of the monetary costs per Instance according to different bidding strategies and task length T. Under the same task length T, the monetary cost followed the *AMAZING* bid strategy is much lower (10 percent) than that conducted by the *HOUR* strategy for various bid prices. The main reason is that in the proposed *AMAZING* strategy, STPM facilitates the decision making by means of predicting the transition among various Instance states and optimizing the balance of reliability versus monetary costs. With intelligence about the Instance state transition, *AMAZING* has more chance to obtain the Spot Instances charged at lower prices by EC2, while keeping a high utilization ratio to complete submitted jobs before deadline.

We also observed that the advantage of *AMAZING* is even more obvious as the task length T increases. As shown in Fig. 9c, the *HOUR* strategy with various bid prices cannot meet the budget constraint of 0.18 USD. In contrast, *AMAZING* reaches budgetary confidence $c_M > 15\%$ even under harsh monetary cost constraint.

In the results of Fig. 9b, the *HOUR* checkpointing strategy with two highest bid prices cannot meet the user's budgetary constraints (0.18 USD), while the three lowest bid prices cannot meet the given deadline constraint (17.6 hours). As we can observe from Figs. 8 and 9, some bid prices followed the *HOUR* strategy are not feasible. The proposed *AMAZING* approach outperforms the *HOUR* strategy in terms of both execution reliability and monetary cost.

5 Related Work

Branches of related work include cloud computing economics and resource management services. Several previous works focus on the economics of cloud computing, i.e., the performance and monetary benefits of cloud computing compared to other traditional computing platforms such as Clusters, Grids, and ISPs [8], [11], [12], [14], [16], [19]. These economic studies are important for understanding the performance tradeoffs among those computing platforms. However, these works assume a static pricing model for EC2's dedicated on-demand instances and do not address the specific and concrete decisions an application scientist must make to balance bid price and resource allocation when using a market-based cloud computing platform, such as Spot Instances. Several systems for monitoring and managing cloud applications exist [3], [4], [6], but these systems currently do not consider cloud prices that vary dynamically over time. Several middleware currently deployed over clouds have fault-tolerance mechanisms [10], [15], but these mechanisms currently are not cost-aware either.

It is a critical challenge to control the balance of reliability versus monetary costs in the context of unreliable resources such as Spot Instances. Probabilistic model [9] and checkpointing mechanisms [20], [21], [22] to answer the question of how to bid given these constraints. Given the maximum price that users are willing to pay per hour, researchers tend to apply probabilistic model and different checkpointing strategies to meet the requirements. Nevertheless, these approaches were considered merely under the fixed-bid price model, and only periodically checkpointing schemes were given in their study. In this work, we try to design an optimal bidding strategy that utilizes both the dynamic pricing model and the state transition intelligence that meets the SLA requirements.

6 Conclusion

In this work, we propose an effective bid decision making strategy to balance the reliability versus monetary costs in the context of unreliable resources such as Spot Instances. Different from previous works on cloud applications, *AMAZING* exploits the intelligence about state transition among various Spot Instances to facilitate decision making during the course of job's computation. Our state context aware design tries to intelligently adapt the bid using intelligence from detected state patterns. In this paper, the decision making optimization problem is formulated as a CMDP. After solving the CMDP, *AMAZING* applies optimal bid decision to each Instance hour until the job's computation is completed. Our experimental results verify that *AMAZING* outperforms previous works in terms of both execution time and monetary cost.

Acknowledgments

Part of the work was published at IEEE Cloud 2012 Conference [18].

References

[1] *Amazon EC2 Spot Instances,* http://aws.amazon.com/ec2/#instance, 2013.

[2] *Amazon Simple Storage Service FAQs,* http://aws.amazon.com/s3/faqs/, 2013.

[3] *CloudKick: Simple, Powerful Tools to Manage and Monitor Cloud Servers,* http://www.cloudkick.com/, 2013.

[4] *CloudStatus,* http://www.cloudstatus.com/, 2013.

[5] http://www.cloudexchange.org/, 2013.

[6] *RightScale: Cloud Computing Management Platform,* http://www.rightscale.com/, 2013.

[7] E. Altman, *Constrained Markov Decision Processes,* vol. 7. CRC Press, 1999.

[8] A. Andrzejak, D. Kondo, and D. Anderson, "Exploiting Non-Dedicated Resources for Cloud Computing," *Proc. IEEE Network Operations and Management Symp. (NOMS '10),* 2010.

[9] A. Andrzejak, D. Kondo, and S. Yi, "Decision Model for Cloud Computing under SLA Constraints," *Proc. 18th Ann. IEEE/ACM Int'l Symp. Modeling, Analysis and Simulation of Computer and Telecomm. Systems,* pp. 257-266, 2010.

[10] J. Dean and S. Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters," *Comm. ACM,* vol. 51, pp. 107-113, 2008.

[11] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The Cost of Doing Science on the Cloud: The Montage Example," *Proc. ACM/IEEE Conf. Supercomputing,* p. 50, 2008.

[12] S. Garfinkel, *Commodity Grid Computing with Amazon's s3 and ec2,* Defense Technical Information Center, 2007.

[13] A. Iosup, O. Sonmez, S. Anoep, and D. Epema, "The Performance of Bags-of-Tasks in Large-Scale Distributed Systems," *Proc. 17th ACM Int'l Symp. High Performance Distributed Computing,* pp. 97-108, 2008.

[14] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. Anderson, "Cost-Benefit Analysis of Cloud Computing versus Desktop Grids," *Proc. IEEE Int'l Symp. Parallel and Distributed Processing (IPDPS '09),* 2009.

[15] M. Litzkow, M. Livny, and M. Mutka, "Condor-a Hunter of Idle Workstations," *Proc. IEEE Eighth Int'l Conf. Distributed Computing Systems,* pp. 104-111, 1988.

[16] M. Plankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon S3 for Science Grids: A Viable Solution," *Proc. Int'l Workshop Data-Aware Distributed Computing (DADC '08),* 2008.

[17] M. Stokely, J. Winget, C. Keyes, and B. Yolken, "Using a Market Economy to Provision Compute Resources across Planet-Wide Clusters," *Proc. IEEE Int'l Symp. Parallel and Distributed Processing Symp.,* 2009.

[18] S. Tang, J. Yuan, and X. Li, "Towards Optimal Bidding Strategy for Amazon Ec2 Cloud Spot Instance," *Proc. IEEE Fifth Int'l Conf. Cloud Computing,* 2012.

[19] W. Wang, B. Li, and B. Liang, "Towards Optimal Capacity Segmentation with Hybrid Cloud Pricing," technical report, Univ. of Toronto, 2011.

[20] S. Yi, J. Heo, Y. Cho, and J. Hong, "Adaptive Page-Level Incremental Checkpointing Based on Expected Recovery Time," *Proc. ACM Symp. Applied Computing,* pp. 1472-1476, 2006.

[21] S. Yi, J. Heo, Y. Cho, and J. Hong, "Taking Point Decision Mechanism for Page-Level Incremental Checkpointing Based on Cost Analysis of Process Execution Time," *J. Information Science and Eng.,* vol. 23, pp. 1325-1337, 2007.

[22] S. Yi, D. Kondo, and A. Andrzejak, "Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud," *Proc. IEEE Third Int'l Conf. Cloud Computing,* pp. 236-243, 2010.

Shaojie Tang received the BS degree in radio engineering from Southeast University, P.R. China in 2006. He received the PhD degree from the Department of Computer Science at Illinois Institute of Technology in 2012. He is an assistant professor in the Department of Computer and Information Science (Research) at Temple University. He has recently served as a guest editor of Journal of *Tsinghua Science and Technology*. His main research interests include wireless networks (including sensor networks and cognitive radio networks), social networks, pervasive computing, mobile cloud computing, and algorithm analysis and design. He also served as a TPC member of a number of conferences such as IEEE ICPP, IEEE IPCCC, MSN. He is a member of the IEEE.

Jing Yuan received the BS degree in computer science from Nanjing University in 2008. She is a graduate student in the Department of Computer Science at University of Illinois, Chicago. Her current research interests include cyber physical systems, online social networks, and cloud computing.

Cheng Wang received the PhD degree from the Department of Computer Science at Tongji University in 2011. Currently, he is with the Department of Computer Science and Engineering, Tongji University. His research interests include wireless communications and networking, mobile social networks, and mobile cloud computing.

Xiang-Yang Li (M'99-SM'08) received the bachelor's degrees from the Department of Computer Science and the Department of Business Management from Tsinghua University, China, both in 1995. He received the MS and PhD degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign in 2000 and 2001, respectively. He is an associate professor of computer science at the Illinois Institute of Technology. He is an editor of several journals, including "*IEEE TPDS" and "Networks.*" He also with the Advisory Board of *Ad Hoc & Sensor Wireless Networks* from 2005, and "IEEE CN" from 2011. He is a senior member of the IEEE.