# AMAZING: An Optimal Bidding Strategy for Amazon EC2 Cloud Spot Instance

ShaoJie Tang*, Jing Yuan†, Xiang-Yang Li*
*Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616
†Department of Computer Science, University of Illinois, Chicago, IL 60607

***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 trade-offs under this schema are of great value for users seeking to lessen their costs while maintaining high reliability. In this paper, we propose a set of bidding strategies to minimize the 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.**

***Keywords*-cloud computing, bidding strategy, EC2.**

## I. Introduction

With the recent advance of Cloud Computing, computing as a utility has been widely recognized and realized. Computing resources can be allocated transparently on an as-need basis. Pricing of these resources can differ dynamically depending on current demand and supply. In December 2009, Amazon released Spot Instances, which is a new way to purchase and consume Amazon EC2 Instances. 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. The Spot Price changes periodically based on supply and demand, and customers whose bids meet or exceed it gain access to the available Spot Instances. For customers with flexibility in when their applications can run, Spot Instances can significantly lower their Amazon EC2 costs. Additionally, Spot Instances can provide access to large amounts of additional capacity for applications with urgent needs. Just a few examples of categories of applications well-suited to Spot Instances are: Image and video processing, conversion and rendering, Scientific research data processing, Financial modeling and analysis.

From above description, there is clearly a trade-off between the reliability of service and cost of the Instances. Existing middleware run on top of these infrastructures fails to cope or leverage changes in pricing or reliability. Ideally, the middleware would have mechanisms to seek by itself the best bidding strategy given the pricing history and demands of the application. In this paper, we investigate bidding strategies that can be used to achieve the goal of minimizing monetary costs under specified reliability requirement, *e.g.*, delay requirement. In specific, a bidding strategy should be able to provide a bidding decision, *e.g.*, the bidding price and time duration, whenever needed. Using real price traces of Amazon's Spot Instances, we study various strategies that can adapt to current Instance Price and show their benefit compared to static, cost-ignorant strategies. Our key result is that the adaptive strategies significantly reduce the monetary cost, while maintaining high reliability.

Notice that when designing a bidding strategy, it is critical to balance the reliability and monetary costs. Previous researchers investigate probabilistic model and checkpointing mechanisms to tackle the problem of how to bid given set of constraints, *e.g.*, cost and reliability constraints. 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 historic data. We first investigate how different prices transit among each other as time goes on. We characterize above transition process using Price Transition Probability Matrix (PTPM) where each entry in the matrix represents corresponding transition probability between two Instance Prices. We show that if the price transition matrix is utilized in an intelligent manner, we can improve the execution efficacy while reducing monetary costs. In this perspective we have designed intelligent bidding strategy *AMAZING*. The novelty of *AMAZING* lies in the intelligence that it learns and adapts from the transition of prices, to make the bid decisions. In specific, in order to make proper bid decision at each Instance hour, we formulated the problem as a Constrained Markov Decision Process (CMDP) [6]. After solving corresponding linear programming, *AMAZING* applies optimal bidding options during the course of the job's computation. In addition to the optimal bidding strategy, we also propose two simple yet efficient heuristic bidding approaches. We are able to show that both heuristic approaches perform very good in terms of high reliability and low monetary cost in practice.

The remainder of this paper is organized as follows. In Section II, we detail the market system of Amazon's Spot Instances, and describe our system model for the application and its execution. In Section III, we present our method for optimizing a user's bid. In Section IV, we show the utility of our model through simulation results. In Section V, we compare and contrast our approach with previous work, and Section VI concludes this paper.

## II. System Model of Spot Instance

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

### A. 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 [15]. Figure. 1 illustrates partial price history for Spot Instance *us-east-1.linux.c1.medium*. 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 exceeds the current Spot Price, her request is fulfilled and her Instances will run until either she chooses to terminate them or the Spot Price increases above her maximum price (whichever is sooner).

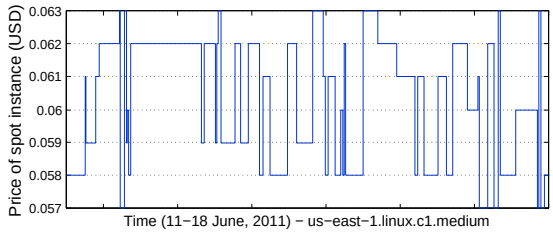


Fig. 1. Spot price fluctuations of *us-east-1.linux.c1.medium* Instance type.

Figure. 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. When a user's maximum price bid exceeds the current Spot Price, the requested resources are granted. Conversely, Amazon stops immediately without any notice when a user's bid is less than the current price. We call this a failure (or out-of-bid). To increase efficiency of task execution, users are suggested to save their intermediate results periodically by means of checkpointing [2].

### B. Problem Formulations

We assume a user is submitting a 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.

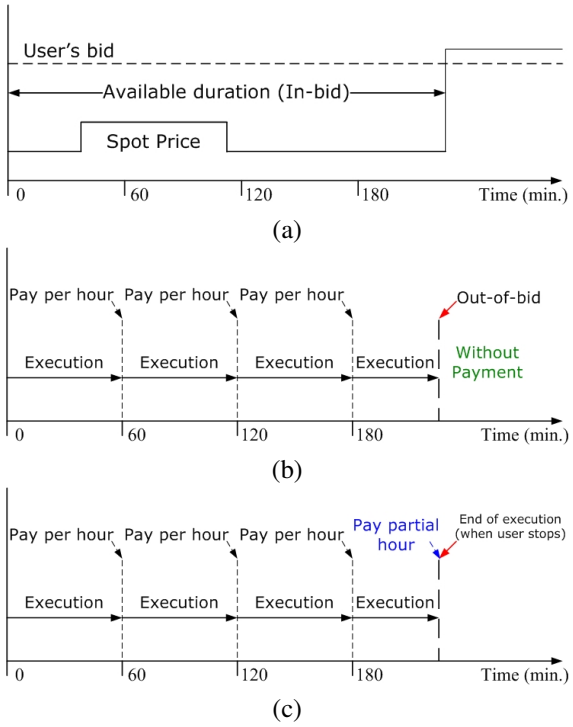


Fig. 2. Examples of pricing for Amazon EC2 Spot Instances: (a) Available duration starts when user's bid exceeds the current Spot Price and ends the other way; (b) When a user's Spot Instance is out-of-bid; (c) When a user stops using Spot Instance.

This is a common type of application that could be submitted on EC2 and amenable to failure-prone Spot Instances [8]. Suppose that the job consists of a total amount of work load $W$ to be executed. We measure $W$ 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 $W$ 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 $T(I_t) = W/C_t$. It is worth to note that $T$ is the net computation time excluding any overheads due to out-of-bid, checkpointing and restart. Therefore $T$ is different from the actual *execution time* $T_e$. Let $t_d$ denote the deadline or delay requirement for a job (for a fixed Instance type), and $E[T_e]$ the expected execution time, our objective is to design a bidding strategy which can minimize the total monetary cost while satisfying delay constraint $E[T_e] \leq t_d$.

In this work, we denote by $t_c$ the time for taking a checkpoint, and $t_r$ the time for a restart. And let $p_{max}$ denote the maximum Spot Price (exclude the burst noises) of a particular Instance type. Further, for a specific Instance type, the Spot Price for the $i^{th}$ 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*, representing whether a job is occupying the computing resource. As discussed later, if the user's current bid exceeds $p_i$, the *result* of the $i^{th}$ hour stays *in-bid*. Otherwise, the *result* becomes *out-of-bid*.

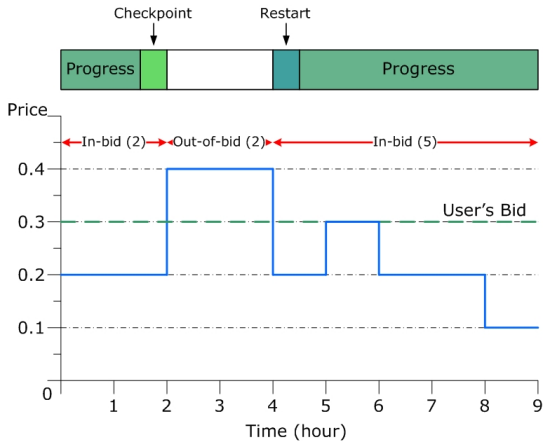


Fig. 3. Illustration of the execution model for Amazon EC2 Spot Instances.

### C. Execution Scenario

Figure. 3 illustrates an exemplary execution scenario. A user submits a job with a total amount of workload $W = 6$ unit-hours (assuming EC2's small Instance server). User's bid price is 0.30 USD, and during the course of the job's computation, the job encounters an *out-of-bid* result (*i.e.* failure) between time 2 and 4. The total available time was 7 hours ($t_c = 0.5$ hour for checkpointing and $t_r = 0.5$ hour for restart), from real execution progress (*i.e.* useful computation) is only $(1.5+4.5) = 6$ hours. The overall execution time $T_e$ is 9 hours. During the job's active execution, the spot price fluctuates: 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, leading to total cost of 1.4 USD. Thus the average cost is $1.4/7 = 0.2$ USD per hour. The per hour average execution progress is $T/T_e = 6/9 \approx 0.67$ hour.

## III. Bidding Strategies

We next propose set of bidding strategies, aiming to minimize the monetary cost under required reliability level. For ease of introduction, we first introduce one simple yet efficient heuristic and then propose our optimal bidding strategy based on Constrained Markov Decision Process.

### A. A Heuristics — Always Bidding Maximum Price

We first propose a heuristic bidding strategy. This heuristic is easy to implement and has good performance in practice. This strategy is called Always Bidding Maximum Price (ABMP). By following ABMP, we always bid the maximum price regardless of current state. The motivation of this strategy can be summarized as follows: Firstly, since ABMP always bid at the maximum price, thus the job Instance will never be terminated by Amazon. In other words, ABMP ensures consistent execution of any job. Therefore the completion time is minimized. Secondly, no matter what strategy we implement, we must pay the minimum price at least. Notice that in current Spot Instance setting, the maximum price is at most 1.1 times the minimum one, thus ABMP will pay at most 10% more than the minimum cost. Therefore ABMP strategy guarantees (1) minimum completion time and (2) at most 10% more than the minimum cost.

### B. AMAZING: CMDP-based Dual-Option Bid Strategy

In this section, we study the following constrained optimization problem: *considering a long state evolving process, given an execution time (delay constraint $t_d$) and price transition probability matrix, what is the optimal bid strategy $\mu$, such that the expected monetary cost $E[M]$ to finish the entire job is minimized, and the expected execution time satisfies $E[T_e] \leq t_d$?*

To tackle the above problem, we design an optimal bidding strategy, namely *AMAZING*, that utilizes both the dynamic pricing model and the state transition intelligence. *AMAZING* is built upon Price Transition Probability Matrix (*PTPM*), a back-end system runs a linear programming routine to select a bid option (make a desirable decision for the next Instance hour) ahead of each Instance hour. The *PTPM* for a particular Spot Instance type can be learnt from Spot Price history released on Amazon EC2 website. With $E$ different Spot Prices for a particular Instance type, we can build an $E \times E$ matrix *PTPM* to represent the transition probability between each Spot Price pair $\sigma$ and $\tau$. In our system, we estimate the transition probability between two Spot Prices $\sigma$ and $\tau$ based on the relative transition frequency. Therefore each entry in *PTPM*, denoted by $\Lambda(\sigma, \tau)$, is defined as the fraction of time when $\sigma$ is followed by $\tau$ in adjacent hours whenever $\sigma$ appears. Clearly, an entry $\Lambda(\sigma, \tau)$ of a higher value indicates a higher transition probability from Spot Price $\sigma$ to $\tau$. In our system, an AI agent is trained to extract Spot Price patterns from the dynamically changed Spot Price data.

Theoretically, a general bidding strategy may bid any price for a specific Instance. However, we prove in Theorem 1 that any bidding strategy can be simulated by so call Dual-Option Strategy where the bidding option in a dual-option strategy is either *bidding the maximum Spot Price* or *bidding zero dollars*.

*Theorem 1:* Any sequence of bidding decisions can be obtained by carefully performing dual-option strategy, either bid the maximum Spot Price or give up, at each instance hour.

*Proof:* Let $p_{max}$ denote the maximum Spot Price (exclude the burst noises) observed from the Spot Price history of a particular type of Spot Instance. Suppose that we have a bidding sequence $\{b_1, b_2, \cdots, b_n\}$. Here, $b_i$ is the bid price for the $i^{th}$ hour. The result for each hour is either *in-bid* or *out-of-bid*, as shown in Figure. 3. If the result of the $i^{th}$ hour is *in-bid* and the Spot Price is $p_i$, then we can substitute $b_i$ by $p_{max}$ *i.e.*, bidding the maximum Spot Price. As a result, the result of the $i^{th}$ hour would not change. Further, since the Spot Instances are charged based on Spot Prices, the user will be charged the same price at $p_i$. On the other hand, if the result of the $i^{th}$ hour is *out-of-bid*, then we can substitute $b_i$ by 0 USD (give up). In this scenario, the result of the $i^{th}$ hour stays the same, and the user will not be charged at all. Thus any bidding strategy, including the optimal one, can be substituted by a dual-option strategy without changing the final output. ■

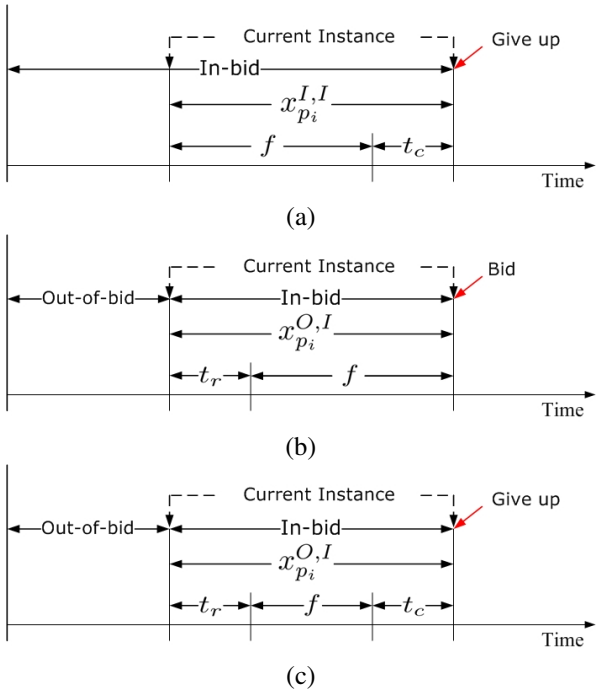


Fig. 4. Examples of execution progress $f$ for Amazon EC2 Spot Instances under different Instance states.

Theorem 1 implies that an optimal dual-option strategy must be also an optimal strategy. Therefore, in the rest of this paper, we can reduce the number of possible bidding options to two instead of infinity without affecting the optimality of our strategy. Let $\mathscr{A}$ denote the set of bidding options that we consider at each hour. Implied by Theorem 1, $\mathscr{A}$ is composed of only two elements: $\mathscr{A} = \{\mathbb{B}, \mathbb{G}\}$. Here $\mathbb{B}$ is short for *bid the maximum price* and $\mathbb{G}$ is short for *bid zero* or *give up*. We denote the bid option made in the $i^{th}$ hour by $\mathbf{a}_i$ where $\mathbf{a}_i \in \mathscr{A}$. The basic idea of *AMAZING* is that during each Instance hour, we pre-select a bidding option, either bid maximum price or giving up, for the next Instance hour with certain selection probability. The selection probability for each bid option is calculated based on current state and PTPM. Since we always make our bid option ahead of time, we can set a checkpoint at current hour whenever we decide to give up in the next hour.

We use $x_{p_i}^{\alpha,\beta}$ to denote an *integrated state* (or simply *state*) of the $i^{th}$ hour. Let $\alpha$ and $\beta$ denote the *result* of the $(i-1)^{th}$ hour and the $i^{th}$ 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, $x_p^{O,I}$ represents such a state: (1) *out-of-bid* in previous hour, (2) *in-bid* at current hour, and (3) the Spot Price at current hour is $p$. By integrating the result from previous hour to current hour, $x_p^{\alpha,\beta}$ provides a compact representation to calculate the *execution progress*. Execution progress represents the actual progress resulted from current ⟨state, bidding option⟩ pair. Let $f(\mathbf{x}_i, \mathbf{a}_i)$ denote the execution progress resulted from state $\mathbf{x}_i$ under bidding option $\mathbf{a}_i$. Refer to Figure. 4 as three examples: (a) assume the result of previous hour is *in-bid*, and current hour is also *in-bid*, then the execution progress is $1 - t_c$ under option *give up*, *e.g.*, $f(x_{p_i}^{I,I}, \mathbb{G}) = 1 - t_c$. This is because it takes $t_c$ time to set the checkpoint at the end of current hour; (b) assume the result of previous hour is *out-of-bid*, and current hour is *in-bid*, then the execution progress is $1 - t_r$ under option *bid*, *e.g.*, $f(x_{p_i}^{O,I}, \mathbb{B}) = 1 - t_r$. This is because it takes $t_r$ time to restart the program at the beginning of current hour; (c) assume the result of previous hour is *out-of-bid*, and current hour is *in-bid*, then the execution progress is $1 - t_r - t_c$ under option *give up*, *e.g.*, $f(x_{p_i}^{O,I}, \mathbb{G}) = 1 - t_r - t_c$. This is because it takes $t_r$ and $t_c$ time to restart the programm and set the checkpoint, respectively. The complete definition of $f(\mathbf{x}_i, \mathbf{a}_i)$ is summarized as follows.

$$f(\mathbf{x}_i, \mathbf{a}_i) = \begin{cases} 1 - t_c, & \text{if } \mathbf{x}_i = x_{p_i}^{I,I}, \mathbf{a}_i = \mathbb{G} \\ 1 - t_r, & \text{if } \mathbf{x}_i = x_{p_i}^{O,I}, \mathbf{a}_i = \mathbb{B} \\ 1 - t_c - t_r, & \text{if } \mathbf{x}_i = x_{p_i}^{O,I}, \mathbf{a}_i = \mathbb{G} \\ 1, & \text{if } \mathbf{x}_i = x_{p_i}^{I,I}, \mathbf{a}_i = \mathbb{B} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

*AMAZING* works as follows: In each Instance hour, *AMAZING* pre-selects a bidding option for the next Instance hour according to decision strategy $\mu$. The calculation of $\mu$ will be discussed in detail later. It is worth noting that during computing the selection strategy $\mu$, we already take into account all potential overhead (*e.g.*, checkpoint, restart). Please refer to Algorithm 1 for pseudo codes.

**Algorithm 1** Pseudo Code for *AMAZING* Policy

**Input**: Optimal bidding strategy $\mu$ (computed from Algorithm 2) and current state information
**Output**: The bidding decision for the next hour

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

In Algorithm 1, $\mu$ specifies the selection probability of each bid option at different states. In following discussions, we focus on finding such an optimal selection strategy $\mu$. We model the computation of selection strategy $\mu$ as Constrained Markov Decision Process (CMDP). By solving the corresponding Linear Programming (LP) in polynomial time [6], we obtain an optimal selection strategy $\mu$ for each state.

*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. To make it fit our problem setting, we define a tuple $\{O, X, \mathscr{A}, P, M, D\}$, where: (a) $O = \{t | t = 1, 2, \cdots\}$ denotes the set of decision epochs. Decisions are made at the beginning of each Instance hour. (b) $X = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$ 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. We have $\mathbf{x}_i = x_{p_i}^{\alpha,\beta}$ for $\mathbf{x}_i \in X$. (c) $\mathscr{A}$ is an action set $\mathscr{A} = \{\mathbb{B}, \mathbb{G}\}$. Theoretically, each Instance hour has $4 \cdot E$ possible states. As defined previously, $E$ 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 $\alpha, \beta$ is 4. Thus we have $4 \cdot E$ possible states for each Instance hour. In order to reduce the searching space, we leverage underlying transition matrix to facilitate our study. We use the previous example in Figure. 4(a) for illustration. In this example, a possible bid option $\mathbf{a}_i$ could be *give up i.e.*, $\mathbf{a}_i = \mathbb{G}$. Notice that current state $\mathbf{x}_i = x_{p_i}^{I,I}$ and bid option $\mathbf{a}_i$ together limit the *valid* state space for the next instance hour to be $\mathbf{x}_{i+1} = x_{p_{i+1}}^{I,O}$, thus the size of searching space for the next instance hour is reduced to $E$. Different bidding options made in the $i^{th}$ Instance hour will lead to different valid state space for the $(i+1)^{th}$ Instance hour. (d) Let $\rho(\mathbf{x}_i, \mathbf{a}_i)$ denote the *occupation measure* of state-option pair $\langle \mathbf{x}_i, \mathbf{a}_i \rangle$, it represents the probability that such state-option pair ever exists in the decision process. Notice that the *occupation measure* $\rho()$ is decided by corresponding decision making strategy. (e) $P$ represents the state transition probability matrix (different from PTPM). Define $P_{i,j}(\mathbf{a}_i)$ as the probability of moving from state $\mathbf{x}_i$ to $\mathbf{x}_j$, when bid option $\mathbf{a}_i$ is taken. Here $\mathbf{x}_j$ denotes any *valid* state of the $(i+1)^{th}$ Instance hour. If $\mathbf{x}_j$ is a *valid* state given $\mathbf{x}_i$ and $\mathbf{a}_i$, then $P_{i,j}(\mathbf{a}_i) = \Lambda(p_i, p_j)$. Recall that $\Lambda(\sigma, \tau)$ is the transition probability from price $\sigma$ to $\tau$. Otherwise, $P_{i,j}(\mathbf{a}_i) = 0$. (f) $M$ 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 Equation. 2. (g) $D$ is the maximum allowed job completion time *i.e.*, deadline. In this paper, we have $D = t_d$. Recall that $f(\mathbf{x}_i, \mathbf{a}_i)$ denotes the *execution progress* for the $i^{th}$ Instance hour. Please refer to Equation. 1 for detailed calculation of $f$ under different states. Therefore, assume the processing capacity is 1 and workload is $W$, we have $E[f] \geq \frac{W}{t_d}$ where $E[f]$, which is calculated in Equation 2, denotes the expected average execution progress per Instance hour.

$$\begin{aligned} E[f] &= \sum_{\mathbf{x}_i \in X} \sum_{\mathbf{a}_i \in \mathscr{A}} \rho(\mathbf{x}_i, \mathbf{a}_i) \cdot f(\mathbf{x}_i, \mathbf{a}_i) \\ E[M] &= \sum_{\mathbf{x}_i \in X} \sum_{\mathbf{a}_i \in \mathscr{A}} \rho(\mathbf{x}_i, \mathbf{a}_i) \cdot p_i \cdot t_d \end{aligned} \tag{2}$$

*2) Optimal Bid Option Selection Strategy $\mu$:* In order to compute the optimal selection strategy $\mu$ 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 [6]. We next write the bid optimization problem defined above as the following LP. The constraints (1) and (3) ensure that $\rho(\mathbf{x}_i, \mathbf{a}_i)$ is a feasible probability measure. The deadline requirement can be respected under constraint (2) by setting the expected average execution progress no less than $\frac{W}{t_d}$. In inequality (4), $\delta_{\mathbf{x}_j}(\mathbf{x}_i)$ is the delta function of $\mathbf{x}_i$ concentrated on $\mathbf{x}_j$, which is defined in following equation: $\delta_{\mathbf{x}_j}(\mathbf{x}_i) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$

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 $\sum_{a_i \in \mathscr{A}} \rho(\mathbf{x}_i, \mathbf{a}_i) \leq 1$, we can not directly use $\rho(\mathbf{x}_i, \mathbf{a}_i)$ as the probability of taking action $\mathbf{a}_i$ at state $\mathbf{x}_i$. Instead, the optimal bid strategy $\mu$ can be determined from $\rho(\mathbf{x}_i, \mathbf{a}_i)$ as follows: $\mu(\mathbf{a}_i|\mathbf{x}_i) = \frac{\rho(\mathbf{x}_i, \mathbf{a}_i)}{\sum_{\mathbf{a}_i \in \mathscr{A}} \rho(\mathbf{x}_i, \mathbf{a}_i)}$. Here $\mu(\mathbf{a}_i|\mathbf{x}_i)$ represents the probability that taking bid option $\mathbf{a}_i$ at state $\mathbf{x}_i$. It is easy to verify that $\sum_{\mathbf{a}_i \in \mathscr{A}} \mu(\mathbf{a}_i|\mathbf{x}_i) = 1$. For any number of input state, Algorithm 2 can return an optimal selection strategy $\mu$ in polynomial time. As input of Algorithm 1, $\mu$ specifies the selection probability of different bid option for different states.

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

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

**Algorithm 2** Computation of Optimal Bidding Strategy $\mu$

**Input**: Execution deadline $t_d$, transition matrix $P$
**Output**: Optimal bidding strategy $\mu$.

1: Solve corresponding CMDP linear programming to get the occupation measure $\rho(\mathbf{x}_i, \mathbf{a}_i)$, $\forall \mathbf{x}_i \in X, \forall \mathbf{a}_i \in \mathscr{A}$;
2: Calculate optimal bidding strategy $\mu$ from $\rho(\mathbf{x}_i, \mathbf{a}_i)$ as:

$$\mu(\mathbf{a}_i|\mathbf{x}_i) = \frac{\rho(\mathbf{x}_i, \mathbf{a}_i)}{\sum_{\mathbf{a}_i \in \mathscr{A}} \rho(\mathbf{x}_i, \mathbf{a}_i)}$$

## IV. 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. 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.

### A. Evaluation Settings

Table. I shows our simulation setup in detail. 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. If not stated otherwise, we use the Instance type D with task length $T$ of 164 minutes (2.7 hours). 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.

TABLE I
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 $\mu$ | $3secs$ |
| 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 |

### B. Impact of Constraint Parameters

In this part, we study how the constraint factors such as budgetary constraint influence 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.

*1) Execution Time and Monetary Cost:* Figure. 5(a) 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 Figure 5(a), 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 $T$.

Figure. 5(b) 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 $10\%$) 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.

*2) Overhead of Checkpoint/restart:* Figure 5(c) 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 IV-C. Clearly, low budgets lead to more frequent out-of-bid situations, which increases the checkpointing overhead.

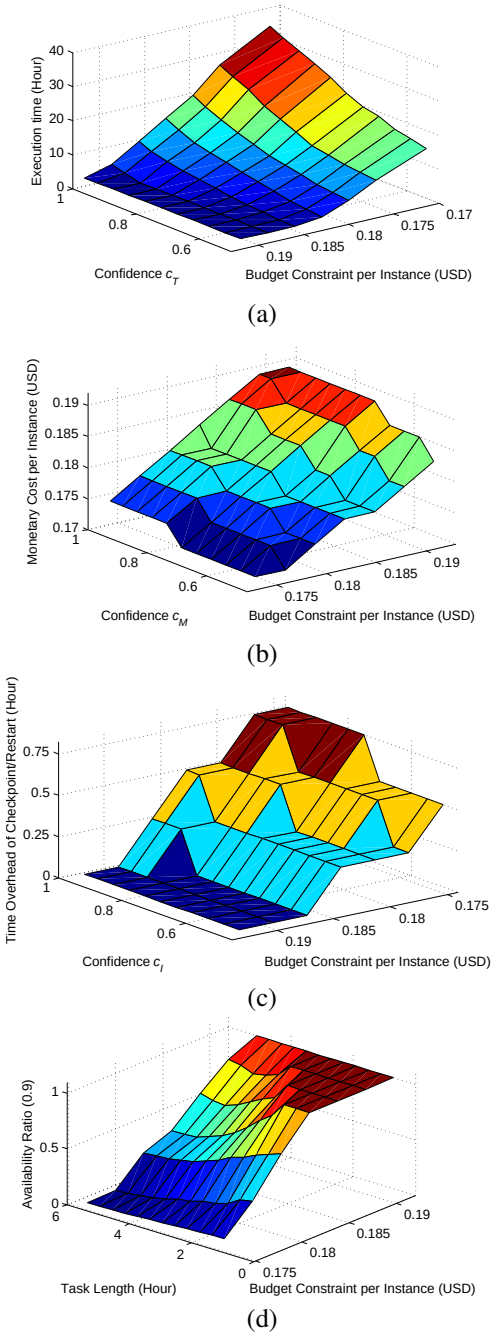


Fig. 5. (a-b)Influence of budget constraint and desired confidences $c_T$ and $c_M$; (c) Overhead of the checkpoint/restart for various budget constraints; (d) Availability ratio depending on the budget constraint and task length $T$.

*3) Influence of Task Length:* Figure 5(d) 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 availability ratio (AR): the ratio of the total time in-bid to execution time *i.e.*, $AR = AT/T_e$. Figure 5(d) shows $AR(0.9)$ *i.e.*, value $v \geq$ ($90\%$ of values assumed by $AR$) as a function of the budgetary constraint and $T$. Here the influence of $T$ is strongly visible,

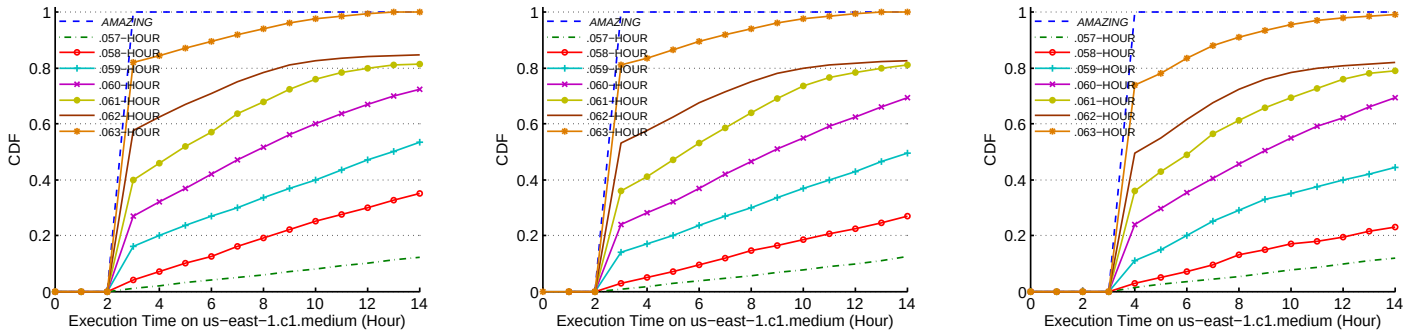


Fig. 6. CDF of execution time for different task length $T$ on Instance type D, where (a) $T = 144$ mins (b) $T = 164$ mins (c) $T = 184$ mins.

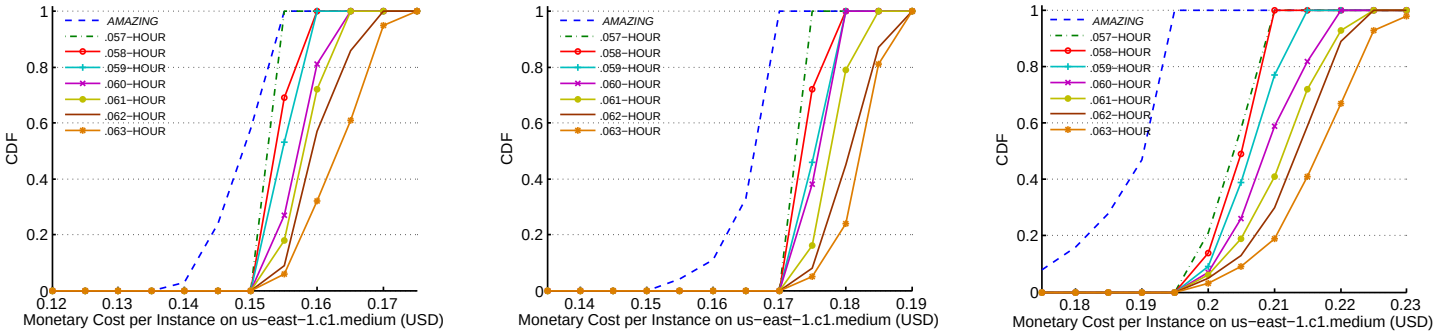


Fig. 7. CDF of monetary cost for different task length $T$ on Instance type D, where (a) $T = 144$ mins (b) $T = 164$ mins (c) $T = 184$ mins.

especially for low budgets. As a consequence, distribution of $AR$ depend on $T$, and cannot be stored only as functions of $T$-*free* factors.

### C. Evaluation of Execution Time

In this section we study performance of *AMAZING* in terms of the execution time compared to previous work [8], [16]. 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 [16] is the hourly strategy, where a checkpoint is taken at a boundary of each paid hour. We conduct extensive experiments to compare the performance of *AMAZING* with the *HOUR* strategy, where the bid price and the task length $T$ are both varied.

Figure 6 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 [16], [8]. Obviously under the same task length $T$, the *AMAZING* strategy has much lower time overhead. We explain this by the fact that our bid option for each Instance hour is computed from the Instance state transition probability matrix (*STPM*). *AMZAING* 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 $20\%$ of the availability time is taken for hourly checkpointing/recovery, less time is utilized for real program processing. In summary, achieving a higher utilization ratio (*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 Figure 6, the $HOUR$ strategy with various bid prices rarely meets the confidence $c_T = 90\%$ under the budget constraint of $0.18$ USD. In contrast, $AMAZING$ reaches $100\%$ confidence of meeting the deadline requirement at less than $4$ hours.

### D. Evaluation of Monetary Cost

In this section we study performance of *AMAZING* in terms of the monetary cost compared to previous work [8], [16]. In Figure 7, we illustrated the cumulative distribution function (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 ($10\%$) than that conducted by the $HOUR$ strategy for various bid prices. The main reason is that in the proposed *AMAZING* strategy, PTPM 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 Figure 7(c), the $HOUR$ strategy with various bid prices can not meets 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 Figure 7(b), the *HOUR* checkpointing strategy with two highest bid prices can not meet the user's budgetary constraints ($0.18$ USD), while the three lowest bid prices can not meet the given deadline constraint ($17.6$ hours). As we can observe from Figure 6 and 7 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.

## V. RELATED WORK

Branches of related work include cloud computing economics and resource management services. Several previous work 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 [7], [10]–[12], [14]. These economic studies are important for understanding the performance trade-offs among those computing platforms. However, these work 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]–[5], but these systems currently do not consider cloud prices that vary dynamically over time. Several middleware currently deployed over Clouds have fault-tolerance mechanisms [9], [13], 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. Previous researchers investigate probabilistic model [8] and checkpointing mechanisms [16]–[18] 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 to meet customers' computing requirements.

## VI. CONCLUSION

In this work, we propose an effective bid option making strategy to balance the reliability versus monetary costs in the context of unreliable resources such as Spot Instances. Different from previous work on cloud applications, *AMAZING* exploits the intelligence about state transition among various Spot Instances to facilitates 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 Constrained Markov Decision Process (CMDP). After solving the CMDP, *AMAZING* applies optimal bid decision to each Instance hour until the job's computation is completed. Our experimental results verifies that *AMAZING* outperforms previous work in terms of both execution time and monetary cost.

## REFERENCES


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

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

[3] *CloudKick: Simple, powerful tools to manage and monitor cloud servers*. http://www.cloudkick.com/.

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

[5] *RightScale: Cloud Computing Management Platform*. http://www.rightscale.com/.

[6] ALTMAN, E. *Constrained Markov decision processes*, vol. 7. CRC Press, 1999.

[7] ANDRZEJAK, A., KONDO, D., AND ANDERSON, D. Exploiting non-dedicated resources for cloud computing, 2010.

[8] ANDRZEJAK, A., KONDO, D., AND YI, S. Decision model for cloud computing under sla constraints. In *2010 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (2010), IEEE, pp. 257–266.

[9] DEAN, J., AND GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.

[10] DEELMAN, E., SINGH, G., LIVNY, M., BERRIMAN, B., AND GOOD, J. The cost of doing science on the cloud: the montage example. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (2008), IEEE Press, p. 50.

[11] GARFINKEL, S. *Commodity grid computing with amazon's s3 and ec2*. Defense Technical Information Center, 2007.

[12] KONDO, D., JAVADI, B., MALECOT, P., CAPPELLO, F., AND ANDERSON, D. Cost-benefit analysis of cloud computing versus desktop grids. IEEE.

[13] LITZKOW, M., LIVNY, M., AND MUTKA, M. Condor-a hunter of idle workstations. In *Distributed Computing Systems, 1988., 8th International Conference on* (1988), IEEE, pp. 104–111.

[14] PLANKAR, M., IAMNITCHI, A., RIPEANU, M., AND GARFINKEL, S. Amazon s3 for science grids: a viable solution. In *Data-Aware Distributed Computing Workshop (DADC)* (2008).

[15] STOKELY, M., WINGET, J., KEYES, C., AND YOLKEN, B. Using a market economy to provision compute resources across planet-wide clusters. IEEE.

[16] YI, S., HEO, J., CHO, Y., AND HONG, J. Adaptive page-level incremental checkpointing based on expected recovery time. In *Proceedings of the 2006 ACM symposium on Applied computing* (2006), ACM, pp. 1472–1476.

[17] YI, S., HEO, J., CHO, Y., AND HONG, J. Taking point decision mechanism for page-level incremental checkpointing based on cost analysis of process execution time. *Journal of Information Science and Engineering 23*, 5 (2007), 1325.

[18] YI, S., KONDO, D., AND ANDRZEJAK, A. Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *2010 IEEE 3rd International Conference on Cloud Computing* (2010), IEEE, pp. 236–243.