Experimental Study of Bidding Strategies for Scientific Workflows using AWS Spot Instances

Hao Wu,
Shangping Ren[*]
Illinois Institute of Technology
10 w 31 St.
Chicago, IL, 60616
hwu28,ren@iit.edu

Steven Timm,
Gabriele Garzoglio[†]
Fermi National Accelerator Laboratory
Batavia, IL, USA
timm,garzogli@fnal.gov

Seo-Young Noh[‡]
National Institute of Supercomputing and Networking,
Korea Institute of Science and Technology Information
Daejeon, Korea
rsyoung@kisti.re.kr

ABSTRACT

Spot instance is an auction based Amazon Elastic Compute Cloud (EC2) instance provided by Amazon Web Service (AWS). It aims to help users to reduce their resource renting cost. The price for spot instances sometimes can be as low as one tenth of the price of the same type on demand instances. However, while gaining significantly cost savings on renting resources, users take risks on running instances without any availability guarantees, i.e. running spot instances can be preempted by Amazon at anytime. Spot instances that get pre-empted are not charged for their last hour and some users utilize that feature to run very short jobs. Different bidding strategies have been proposed to ensure the execution performance of tasks submitted to spot instances. In this paper, we present a full EC2 spot instance simulator that uses real EC2 spot pricing history to emulate the spot instance life cycle and expected charges. We review eight of the most popular bidding strategies in both literature and practice and compare them in terms of cost, deadline miss rate and task's execution length for scientific workflows. Our evaluation provides users a guidance on how different bidding strategies may impact the execution of scientific workflows.

1. INTRODUCTION

Amazon Web Service (AWS), an infrastructure-as-a-service (IaaS) cloud provided by Amazon is one of the largest public cloud service providers. The increasing popularity of utilizing cloud computing services is driven by two fundamental merits provided by cloud services: elastic and economic. With the elastic IaaS services, users can acquire both computing and storage resources as needed and only need to pay for the resources when they use the resources. Hence, with cloud services, users not only save monetary cost but also save the time and efforts on building computing infrastructures.

Because of these advantages, increased number of companies and organizations have started migrating their existing compute infrastructures to computer clouds. According to Google, 95% of web services are now deployed on cloud [5]. It is estimated that the cloud service market will grow to \$270B by 2020 worldwide [6]. The cloud advantages also attract researchers to utilize cloud for scientific computing. For instance, large research institutes such as Fermi National Accelerator Laboratory(Fermilab) [14], Argonne National Laboratory (ANL) [9], Brookhaven National Laboratory (BNL) [4], CERN [7] and others have begun executing scientific workflows on computer clouds.

However, there are still challenges yet to be solved for deploying scientific workflows on computer clouds. Scientific workflows are distinguished from general purpose workflows by the large amount of computing resources and execution time that it takes to complete the scientific workflow, in addition to the large amounts of data that are processed by the scientific workflow. Hence, although cloud is cost-effective in general, it can be costly for large size scientific workflows.

In later 2009, Amazon provided an auction based instance type – spot instance which allowed users to bid unused EC2 resources. Spot instances can be available at a cost as low as one tenth of the regular on-demand prices. Sometimes, users can even get free instances for short jobs. The essence behind such a huge price difference is that the service provider (Amazon) is willing to fully utilize their resources by selling spare unused resources in a relatively low price. However, while gaining significant cost reduction on renting resources, users also take risks on running instances without guarantees, i.e. spot instances may be preempted during their execution by Amazon at any time. Hence, many researchers have studied the trade-offs between cost and spot instance availability. Different bidding strategies have been proposed

[*]The research is supported in part by NSF under grant number CAREER 0746643 and CNS 1018731.

[†]This work is supported by the US Department of Energy under contract number DE-AC02-07CH11359

[‡]This work is supported by KISTI under a joint Cooperative Research and Development Agreement CRADA-FRA 2015-0001 / KISTI-C15005.

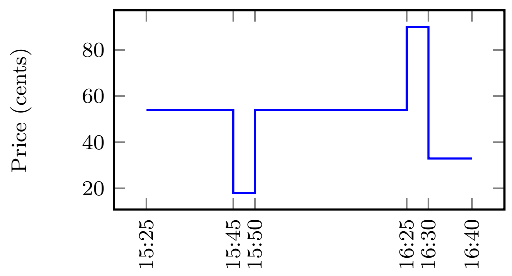

Figure 1: Spot Price Variation for m3.2xlarge Instance (2015-4-30 15:25 – 2015-4-30 16:40)

to ensure successful execution of submitted tasks and at the same time minimize of the total execution cost.

However, different bidding strategies have their own assumptions and use cases. Choosing a bidding strategy with the most balanced performance for scientific workflows is a challenging task. In this paper, we first develop a full EC2 spot instance simulator that emulates EC2 spot instance ecosystem based on real EC2 spot price history, i.e. spot instance's life cycle and charging behaviors. We review eight of the most popular bidding strategies in the literature and from practice. We evaluate their performance in terms of cost, deadline miss rate and task's execution length through large numbers of simulations. Our evaluation conclusions provide users a guidance on how different bidding strategies may impact the execution of scientific workflows.

The rest of paper is organized as follows: Section 2 describes the EC2 spot instance ecosystem. Different bidding strategies are reviewed in Section 3. Detailed design of our evaluation is presented in Section 4. Section 5 shows the evaluation results. At last, we conclude our work in Section 6

2. AWS SPOT INSTANCE

The AWS Elastic Compute Cloud (EC2) spot instance allows users to bid their own price to rent the instance. If a user wins the bid (larger or equal to market price), the user's spot instances will be instantiated. If the user loses the bid (bid price is lower than market price), the user's running spot instances are terminated by Amazon automatically.

2.1 Bidding

It is not difficult to see that the key behind the spot instance is the bidding. As long as the bid is always above market price, user's spot instances will continue running without interruption. Although a user can bid at an extremely high price for spot instances to keep the instances running, once the market price exceeds the on-demand price, renting spot instance costs more than renting on-demand instances. In order to prevent unreasonable bid and extremely high cost, Amazon allows a maximum bid of ten times the on-demand price for each instance type [1].

There are two types of spot instances a user can bid. One is called *one time* spot instance and the other is *persistent instance*. For one time instance, the instance will be instantiated once when the bid exceeds the market price and the instance will be terminated by Amazon when the bid drops below market price or terminated by the user. Different from one time instance, for the persistent bid, spot instances remains in the system after being terminated by Amazon and be re-instantiated once the bid surpass the market price again until they are terminated by users. It is worth noting that, for both types of spot instances, once a bid is made, it cannot be changed during the VM instance's life time.

Amazon also support "launch group" bid, for which users can make a single bid for a group of same type spot instances. However, with the group bid, once one member of the launch group is terminated by Amazon, the entire group of the instance are terminated. In general case, Amazon only allows 20 active spot instance bids for each account in each region. For large amount of spot instance requests, user can use Spot Fleet [2] to bid spot instances which can get as much as 1000 spot instance bids per region.

2.2 Market Price

A bid one user made only presents the maximum price the user is willing to pay rather than the actual price the user pays for renting spot instances. The actual cost a user payed for spot instances is determined by market price. The market price is decided based on the total bids Amazon received and the size of the spot pool in a region. Amazon sorts all received bids in a decreasing order, and use the size of the spot pool to find the cut-off price. Such cut-off price is called *market price* [1]. All the bids above the cut will be granted spot instances. Once a new bid received, Amazon re-sorts the bids and then a new market price is determined.

Amazon keeps the records for all the market prices within the latest ninety days so that users are aware of recent price changes for spot instances. Such record is called Spot Pricing History, and can be retrieved from both AWS management console and APIs. Figure 1 illustrates a spot pricing history from 2015-04-30 15:25:00 to 2015-04-30 16:40:00 for m3.2xlarge instances in us-west-2a region. As shown in the figure, within the 2 hour window, the market price changed four times.

2.3 Charging

Amazon charges spot instance in integral hours starting from the time instance that a user's spot instance is granted. For each integral hour, Amazon charges market spot price at the beginning of that integral hour. If the spot instance is terminated by Amazon, there is no cost for that integral hour. However, if the spot instance is terminated by user, Amazon charges the entire integral hour. Take Fig. 1 as an example. If a user bids one dollar at time 15:25, because the bid is larger than the market price (53 cents), the spot instance is granted and will start running. When the market price drops to 17 cents at 15:45, as the bid is still larger than the market price, the instance keeps running. As the price change occurs within one hour after the bid is granted, the first hour cost is based on the market price at the beginning of that hour which is 53 cents. When the instance is running for the second hour, the market price changes to 90 cents at the beginning of the second hour. Hence, the cost for the second hour is charged at 90 cents.

2.4 Preemption Notification

Starting from early year 2015, Amazon provides a new service to notify users the termination of their spot instances [8]. However, the notification is only available two minutes before the termination and only can be retrieved from inside the spot instance. We have not yet implemented a way to respond to the notification within the 2-minute time window.

3. BIDDING STRATEGIES OVERVIEW

Based on the spot instance ecosystem, it is not difficult for us to find out that different bidding strategies can result in different costs on renting spot instances. For instance, if a user bids at a low price, it is guaranteed low cost for running spot instances. However, the availability of the spot instance is not guaranteed. Many bidding strategies are proposed in the literature to balance the trade-offs between cost and availability. They can be categorized in two classes: static bidding and dynamic bidding.

3.1 Static Bidding Strategy

Static bidding strategy is to bid a constant price. It does not change as the market price changes. The advantage of static bid is it is simple to implement. The drawback of the static bid is also obvious: it may not obtain any spot instance.

One typical example of using static bidding strategy is ATLAS team in BNL which always bid with one quarter of the on-demand price [10]. In this paper, we also examine other static bidding strategies such as bid with on-demand price, bid with the maximum price in the spot pricing history, bid with the absolute maximum price (ten times of on-demand price), bid with the minimum price in the spot pricing history, bid with 25% more of the minimum price in the spot pricing history.

3.2 Dynamic Bidding Strategy

Different from static bidding strategy, dynamic bidding strategy dynamically adjusts bid prices according to application's execution requirements and market prices.

In 2012, Song *et al.* proposed an optimal bidding strategy for cloud service broker [12]. They first deconstruct the spot pricing history data and model the market prices using semi-Markovian chain, and formulate the problem as a cloud service broker profit maximization problem. To solve the problem, they design a profit aware dynamic bidding algorithm to calculate the optimal bid that maximizes the profit for cloud service brokers.

Song *et al.* also proposed an optimal bidding strategy for deadline constrained jobs [15]. The optimal bidding strategy calculates the bid according to the probability distribution of all the market prices existed in the spot pricing history and the remaining deadline at the beginning of each instance hour. In their paper, the authors assume that market prices are uniformly distributed. In order to meet application's deadline, once the remaining execution time equals to the deadline, the application is immediately migrated to on-demand instance.

Tang *et al.* also proposed an optimal bidding strategy for deadline constrained jobs [13]. The authors first theoretically proved that the optimal bidding strategy can be covered by a dual-option strategy: either bid with the maximum price or with zero. Then authors build a Price Transition Probability Matrix (PTPM) that records the probability of price changes from one to another in the spot pricing history. Based on the PTPM, they formulate the problem of minimizing the cost under required reliability level as a Constrained Markov Decision Process (CMDP). An optimal bid that minimizes the cost is obtained by solving the problem through linear programming, .

Recently, Bogumil *et al.* proposed an adaptive bidding strategy that minimizes the cost for renting spot instances [11]. They first find the minimal price per ECU across all instance types and availability zones from the spot pricing history. Their adaptive bidding algorithm is to find the current cheapest per ECU instances across all the instance types and availability zones. If the current lowest price is above a predefined threshold, the algorithm withdraws the bid. Otherwise, it bids with the lowest price. In addition to find the lowest bid, the adaptive algorithm also calculate the checkpointing frequency for the application.

Although some bidding strategies are claimed as optimal bidding strategies, their conclusions are based on their specific assumptions and application scenarios. In this paper, we investigate the applicability of these strategies in scientific workflow settings. In particular, we compare a set of static and dynamic bidding strategies through simulation using real Amazon spot pricing history data and real Fermilab scientific workflow requirements settings. The comparisons are performed against different evaluation criteria. The detailed simulation design and evaluation criteria are presented in next section.

4. AWS SPOT INSTANCE SIMULATOR DESIGN

In order to evaluate the performance of different bidding strategies, we implement a EC2 spot instance simulator that can fully emulate AWS spot instance running status and charging behavior based on real spot pricing history. The simulator is able to automatically retrieve spot pricing history from Amazon and store the data in a local database. In this manner, we can keep more than 90 days of real spot pricing data. The simulator supports both *one time* and *persistent* spot instances. The inputs of the simulator are user's bid price and bid time. The simulator calculates the instance start time, running duration and total cost. The simulator is written in Python.

4.1 Simulation Design

We first give definitions for the terminologies used in the simulation design.

Job: we model a job as a two-tuple $j(e, D)$, where e is the execution time demand and D is the deadline of the job, respectively. Both execution time demand and deadline are in instance hours.

Success Bid: if a bid is higher or equal to the market price, it is a successful bid. Otherwise it is an unsuccessful bid. A successful bid doesn't guarantee successful execution of a task.

Failed Execution: if with a successful bid, a job cannot finish its execution (preempted by AWS due to price change), it is counted towards a failed execution. If with a successful bid, a job can finish its execution, but its finish time exceeds deadline, it is also counted towards a failed execution.

Successful Execution: if with a successful bid, a job finishes its execution before deadline, it is counted towards a successful execution.

An Execution: an execution E is represented as a 3-tuple (b, p, d), where b is the bid price, p is market price of the time the bid is made and d is instance running duration with the bid. The duration d is in integral instance hours. If $d < e$, then E is a failed execution. If $d \geq e$, then E is a successful execution.

Simulation (s): one simulation s is to bid spot instance to finish a job j within the job's deadline D. In one simulation, the simulator always tries to bid for instances after unsuccessful bids or failed executions. The simulation terminates when a job is successfully executed or it fails by missing deadline. The bid frequency after unsuccessful bids and failed executions is one hour.

Failed Execution Set ($F(s)$): the failed execution set of one simulation is defined as $F(s) = \{E_1, \ldots, E_n\}$.

Successful Execution Set ($S(s)$): the successful execution set of one simulation is either an empty set or a set only contains one element(a successful execution).

Deadline Miss($d(s)$): if the successful execution set of one simulation is empty, then we say the job misses its deadline. Hence, $d(s) = 1$ if $S(s) = \emptyset$. Otherwise, $d(s) = 0$.

Immediate Start (IS(s)) : if the first bid is a success bid, $IS(s) = 1$, otherwise, $IS(s) = 0$.

Cost ($C(s)$): the total cost to execute a job includes all the cost of failed executions and the successful execution. Hence, the total cost is defined as:

$$C(s) = \sum_{i=1}^{|F(s)|} (p_i \times d_i | b_i \in F(s)) + \sum_{i=1}^{|S(s)|} (p_i \times d_i | b_i \in S(s)) \tag{1}$$

4.2 Evaluation Criteria

For one instance type on one U.S. availability zone, we run the single simulation every one hour based on the spot pricing history data. To evaluate the performance of different bidding strategies, we compare statistics of simulation results from two aspects. One is whether jobs finish execution within their deadlines. The other is the cost to execute the jobs. To be more specific, we define $\text{Sim} = \{s_1, \ldots, s_n\}$ as the set of simulations performed in one availability zone for a given instance type.

Immediate Start Rate (ISR) measures the possibility that a spot instance can be granted with the first bid:

$$\text{ISR} = \frac{\sum_{i=1}^{n} \text{IS}(s_i)}{n} \tag{2}$$

Deadline Miss Rate (DMR) measures the possibility that a job misses its deadline:

$$\text{DMR} = \frac{\sum_{i=1}^{n} d(s_i)}{n} \tag{3}$$

Average Cost to Demand Rate (ACDR) measures the average cost on executing a job using spot instance compared with the cost on executing a job using on-demand instance. It is defined as follow:

$$\text{ACDR} = \frac{\sum_{i=1}^{n} C(s_i)}{n \times e \times p_d} \tag{4}$$

where p_d is the on-demand price for the given instance type.

Expected Execution Length to complete a job (EEL) estimates the average total execution time required to finish a job. It is defined as follow:

$$\text{EEL} = \frac{\sum_{i=1}^{n} (E(s_i) \times (1 - d(s_i)))}{\sum_{i=1}^{n} (1 - d(s_i))} \times \frac{1}{1 - D(\text{Sim})} \tag{5}$$

where $E(s_i)$ is the execution length of i^{th} simulation.

5. EVALUATION RESULTS

The evaluation results are based on the simulation on almost four months real EC2 spot pricing history data from 2015-04-03 to 2015-07-29. We run simulations for all the m3 and c3 instances across all the U.S. regions and availability zones. We evaluate the performance of bidding strategies on four different jobs settings, i.e. $j_1(5, 168), j_2(10, 168), j_3(24, 168)$ and $j_4(100, 168)$. In total, we total run 324 sets of simulations. On average, each set of simulations contains more than 2000 distinguished single simulations. We evaluate eight different bidding strategies, i.e. bidding the minimum price in the spot pricing history (MinPrice), bidding 25% more of the minimum price in the spot pricing history (Min+25), bidding the maximum price in the spot pricing history (MaxPrice), bidding the on-demand price (DemandPrice), bidding the absolute maximum price (Demandx10), bidding quarter of the on-demand price (Demandx.25), bidding with adaptive bidding strategy (AdaptiveBid) [11] and bidding with the optimal bidding strategy (OptimalBid) [15].

Due to the page limit, we only illustrate the evaluation results for m3.2xlarge instances on us-west-2a availability zone.

Figure 2 depicts the evaluation results for executing 5-hour jobs using m3.2xlarge instance on us-west-2a availability zone. The figure shows that no job is able to finish its execution when bidding with the minimum price in the spot pricing history. However, if we raise the bid to 25% more of the minimum price, the performance is significantly improved. Almost 75% of the jobs are able to finish before their deadlines. Bidding with quarter of on-demand price results in the similar performance as the Min+25 strategy as they result in almost at the same bidding price. If we raise the bid to the on-demand price, the job success execution rate can reach 90%. However, it will cost 9% more on all jobs' execution and 7% more on all success jobs' execution.

Since the absolute maximum bidding price is larger or equal to the maximum price exists in spot pricing history, both of them guarantee immediate start of spot instances and all the jobs finish their execution within the deadlines. However, the average cost for completing all jobs reaches 35% of the on-demand prices which is almost the double of the cost when bidding with on-demand price.

Surprisingly, the adaptive bidding strategy performs not as well as we would have expected. Almost half of the jobs

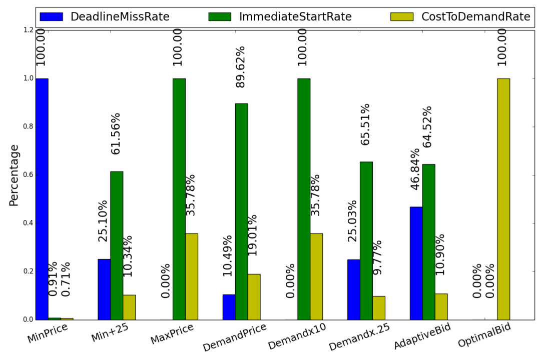

Figure 2: Performance Comparison for 5-Hour Job

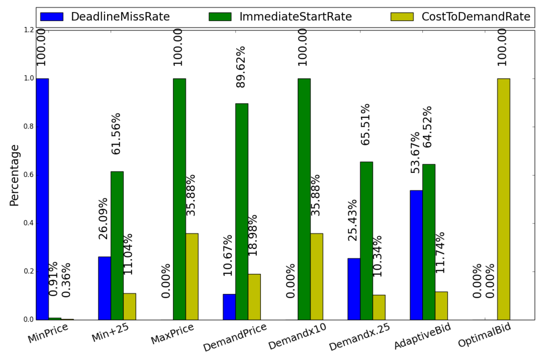

Figure 3: Performance Comparison for 10-Hour Job

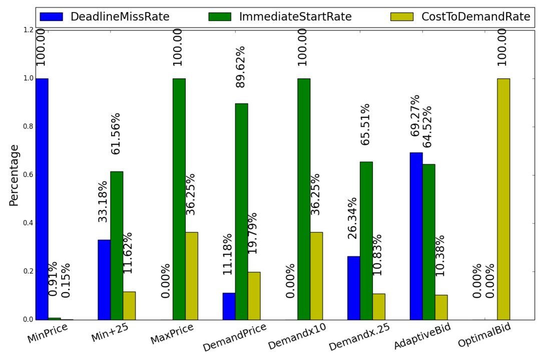

Figure 4: Performance Comparison for 24-Hour Job

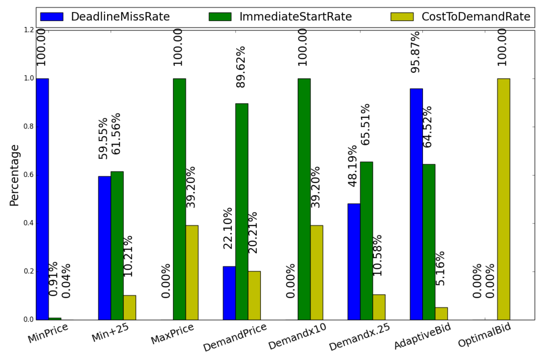

Figure 5: Performance Comparison for 100-Hour Job

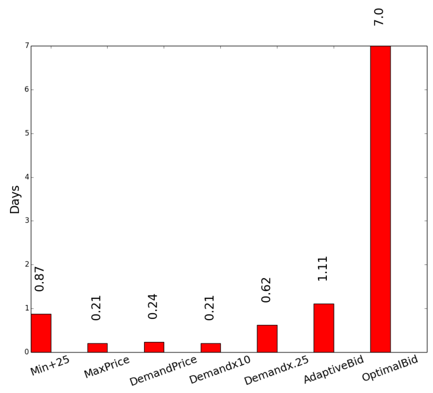

Figure 6: Average EEL for 5-Hour Jobs

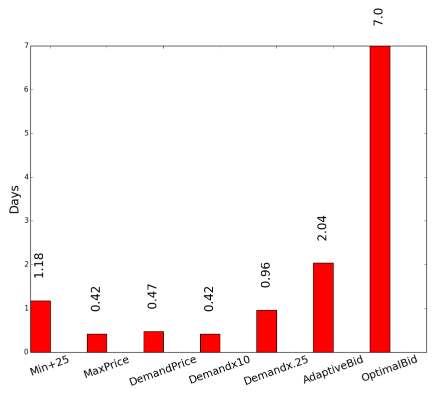

Figure 7: Average EEL for 10-Hour Jobs

failed execution with adaptive bidding. The reason that adaptive bidding strategy has such a high deadline miss rate is that the adaptive bidding algorithm withdraws its bid when the calculated bid exceeds the predefined threshold. Since the adaptive bidding algorithm seeks the cheapest instance, it withdraws many bids and leads to high deadline miss rate.

With the optimal bid, all jobs are able to finish within deadline. However, if we look into the details of optimal bid, we find that the optimal bid strategy can hardly get spot instances and all the jobs are actually executed using on-demand instances. The major reason that optimal bid cannot obtain any spot instance is that the optimal bid assumes all prices in the price history are follow uniform distribution which is apparently not the case in practice.

Fig. 3, Fig. 4 and Fig. 5 show the evaluation results for executing 10-hour jobs, 24-hour jobs and 100-hour jobs using m3.2xlarge instance, respectively. As the execution demand increases, both deadline miss rate and the average cost for executing jobs increases. Figure 6, Fig. 7, Fig. 8 and Fig. 9 depict the average expected execution length for 5-hour jobs, 10-hour jobs, 24-hour jobs and 100-hour jobs, respectively. Since when bidding with minimum price can never get jobs to finish, we eliminate the minimum bid from the figures. By reviewing all the evaluation results, we can conclude that bidding with quarter of on-demand price gives most balanced performance in terms of cost, deadline miss rate and expected job execution length for scientific workflows.

The complete evaluation results for all m3 and c3 instances on all U.S. availability zones can be found in our

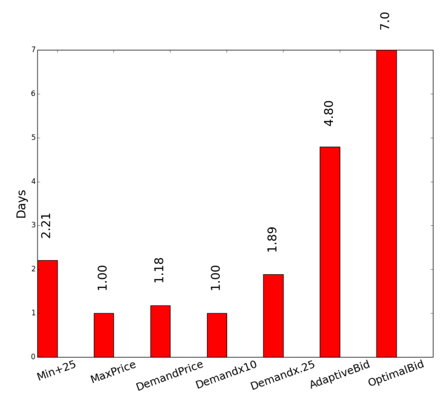

Figure 8: Average EEL for 24-Hour Jobs

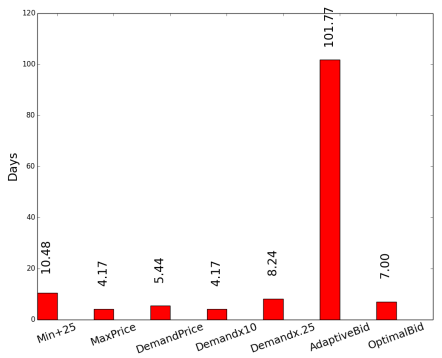

Figure 9: Average EEL for 100-Hour Jobs

website [3].

6. DISCUSSION AND CONCLUSION

In this paper, we study the Amazon EC2 spot instance ecosystem and experimentally evaluated eight most popular bidding strategies in the literature. We develop a simulator that can fully emulate the spot instance running status and charging behavior. We use the simulator to evaluate the eight different bidding strategies in terms of job execution cost, job deadline miss rate and expected job execution length through large simulation runs. Through the evaluation, we conclude that bidding with 25% of on-demand price give most balanced performance for scientific workflows.

In addition, we also find that all the dynamic bidding algorithms do not perform as well as we would have expected. Our study reveals that these dynamic strategies are based on some assumptions which do not hold in reality. For instance the assumptions that a bid can be changed after the bid is made [15]; checkpointing can be perfectly performed before spot instances get preempted by Amazon [13]; are not valid in practice.

Another assumption that is made by all the literature [15, 13, 11] is that a single bid will not affect the spot pricing. However, as previously mentioned, scientific workflows require large amounts of resources which makes the assumption invalid. Large amount of spot instance requests can have significant impact on the entire spot market. Hence, it is possible that none of the current bidding strategies works well for large amount spot instance biddings. One of our future work is to study the impact of large scale bids and the performance of different bidding strategies on large scale bids. Since we only evaluate performance of bidding strategies on single instance types in single availability zone in our current work, another future work is to explore the bidding strategies for finding the cheapest spot instances and that are least likely to be pre-empted across multiple availability zones.

7. REFERENCES

[1] AWS EC2 Sport Instance. `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html`.

[2] AWS EC2 spot fleet. `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet.html`.

[3] CODE. `code.cs.iit.edu`.

[4] Feature - clouds make way for STAR to shine. `http://www.isgtw.org/feature/isgtw-feature-clouds-make-way-star-shine`.

[5] Google cloud platform roadshow 2014. `https://speakerdeck.com/googlecloudplatform/keynote-cloud-developer-roadshow-2014`.

[6] http://www.marketresearchmedia.com/?p=839.

[7] Mapping the secrets of the universe with google compute engine. `https://cloud.google.com/developers/articles/mapping-the-secrets-of-the-universe-with-google-compute-en`
`hl=ja`.

[8] New EC2 spot instance termination notices. `https://aws.amazon.com/blogs/aws/new-ec2-spot-instance-termination-notices/`.

[9] Nimbus and cloud computing meet STAR production demands. `http://www.hpcwire.com/hpcwire/2009-04-02/nimbus_and_cloud_computing_meet_star_production_demands.html`.

[10] M. Ernst, C. Gamboa, J. Hover, H. Ito, and M. OConnor. Running ATLAS at scale on amazon spring 2015 HEPiX. 2015.

[11] B. Kamiński and P. Szufel. On optimization of simulation execution on amazon ec2 spot market. *Simulation Modelling Practice and Theory*, 2015.

[12] Y. Song, M. Zafer, and K.-W. Lee. Optimal bidding in spot instance market. In *INFOCOM, 2012 Proceedings IEEE*, pages 190–198. IEEE, 2012.

[13] S. Tang, J. Yuan, and X.-Y. Li. Towards optimal bidding strategy for amazon EC2 cloud spot instance. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 91–98. IEEE, 2012.

[14] S. Timm, G. Garzoglio, S. Fuess, and G. Cooper. Virtual facility at fermilab: Infrastructure and services expand to public clouds. In *The International Symposium on Grids and Clouds (ISGC)*, volume 2015, 2015.

[15] M. Zafer, Y. Song, and K.-W. Lee. Optimal bids for spot vms in a cloud for deadline constrained jobs. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 75–82. IEEE, 2012.