Keep It Simple: Bidding For Servers In Today's Cloud Platforms

Prateek Sharma, David Irwin, Prashant Shenoy

Abstract—Public clouds now offer computing services with a variety of pricing schemes. Spot servers, which offer large cost savings, are an increasingly popular platform on which applications are being deployed. Spot servers are dynamically priced and require users to submit a bid. In this article, we show the effect of bidding on application cost and availability when running on spot servers. Based on our analysis, we present simple and effective bidding strategies for users and motivate new research directions in cloud resource management and fault tolerance.

1 Introduction

Today's IaaS cloud platforms such as Amazon EC2 and Google Cloud Platform rent computing resources on demand in the form of virtual machine servers and offer numerous benefits, including a pay-as-you-use pricing model, the ability to quickly scale capacity when necessary, and low costs due to their high degree of statistical multiplexing and massive economies of scale.

IaaS platforms rent servers under a variety of contract terms that differ in their cost and availability guarantees. The simplest type of contract is for an *on-demand* server, which a customer may request at any time and incurs a fixed cost per unit time of use. In contrast, *spot* servers provide an entirely different type of contract for the same resources. Spot servers incur a *variable* cost per unit time of use, where the cost fluctuates continuously based on the spot market's instantaneous supply and demand. Unlike on-demand servers, spot servers are *revocable*: the cloud platform may unilaterally preempt them at any time.

In the case of Amazon EC2, the cost and availability of spot servers is governed by an auction mechanism. A customer specifies an upper limit (a "bid") on the price they are willing to pay for a spot server, and EC2 reclaims the server whenever the server's spot price rises above the bid. Since spot servers incur a risk of unexpected resource loss, they offer weaker availability guarantees than on-demand servers and tend to be cheaper—the average price of spot servers is only 10-30% of the on-demand servers.

Conventional wisdom has held that careful selection of bid-price is important to balance the cost-availability tradeoff—since a high bid may increase costs but also increase spot server availability. In this article, we show that spot instance bidding need not be complicated. We analyze empirical price data of over 1,500 spot markets over a six month period, and show that a wide range of possible bids have approximately the same intended effect on the cost and availability. We show that while careful bid selection doesn't significantly impact the cost-availability tradeoff, careful *spot market selection* is important to reduce costs and effects of revocations.

Based on our analysis, we argue for simple bidding strategies and describe best-practices when deploying applications on spot servers. We identify challenges and opportunities in reducing the impact of spot revocations (which are akin to machine failures) on application performance. Our goal is to provide practical suggestions to simplify bidding, and to motivate new directions in cloud computing research.

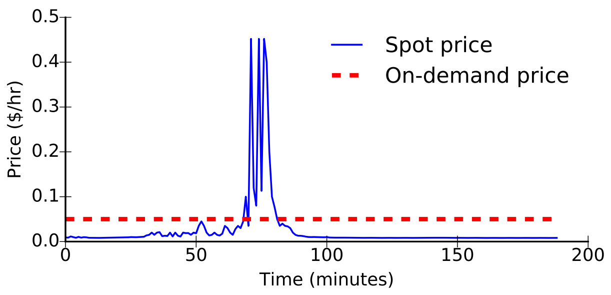

Fig. 1. Variations in spot price of the m3.medium instance type. The spot price is generally much lower than the on-demand price, but shows ocassional spikes.

2 Bidding For Spot Instances

Spot instances allow cloud platforms to gain revenue from surplus idle resources. Amazon EC2 uses a market mechanism to sell this capacity where users place a bid for servers, and EC2 allocates them if the bid is higher than the spot price, which varies continuously based on supply and demand. When the spot price rises above a user's bid price, EC2 revokes the servers. EC2 determines the spot price by running a sealed-bid multi-unit second-price auction [2]. Note that the underlying supply of surplus servers in the spot pool also changes dynamically, since EC2 may take resources from the spot pool to allocate new on-demand instances. Thus, the spot price changes dynamically both as users submit new bids, and as the spot pool's capacity changes (Figure 1).

To use a spot server, users place a single, fixed bid, which represents the maximum hourly price that they are willing to pay. The bids can range from zero to ten times the on-demand price. Based on the current bids for the server and the available supply, a spot price is determined by a continuous auction. Because this is a second-price auction, users pay the *spot price*, which may be lower than the bid. If the market price increases above the user's bid, then the spot instance is revoked and terminated after a small (120 second) warning. The prices for each spot server type (also referred to as a *spot market*) are independently determined. The combination of different server sizes and geographical regions determines a market, and Amazon runs over 2,500 spot markets globally.

A low bid means that the user is price-sensitive and is only willing to pay a low price for the spot servers. But a server with a low bid may suffer from low availability and higher likelihood of being revoked if the market price increases above the bid price. Frequent revocations may cause application downtimes, missed deadlines, and decreased performance as the application recovers from revocations which are akin to machine failures. Thus bidding presents the user with a tradeoff between cost and availability/revocation-rate, which may further impact application performance.

Careful selection of bids via bidding strategies has received wide attention in both research [7] and industry. Bidding strategies have been proposed for minimizing costs with different constraints (deadlines, etc.) for a wide range of applications (Map-Reduce, scientific-computing, etc.). Bidding's complexity may be one reason why, despite its extremely low prices (70-90% less than on-demand instances), the spot market has low utilization [3]. As we discuss, however, the bidding problem in today's markets (and possibly in future markets) is not particularly important for maximizing performance and minimizing costs using spot servers.

3 Effect of Bidding

To understand the effect of bidding for spot instances, we analyze spot prices over a six month period from March to August 2015 (and longer periods where stated) of 1,500 spot markets. For ease of exposition, we begin our discussion by analyzing the most popular instance types in the most popular region, i.e., Linux instances in the us-east-1 region.

Bidding strategies optimize the cost-availability tradeoff for spot instances: as a user increases their bid, they may pay more per-hour, but their availability also increases. However, spot price data across many markets shows that there is a wide range of "optimal" bids that essentially yield the same availability for the same cost. This is because the spot prices are "spiky". In figure 1 we see that the price spikes can be almost 10× the on-demand price—the same as the upper-bound on the bid-price. Thus no matter what the bid, the spot instance will be revoked during these large spikes.

To illustrate, Figure 2(a) shows a CDF of availability for instance types in five different markets over our six month period, where the x-axis is a user's bid normalized to the on-demand price, i.e., 2 is 2× the on-demand price, etc. As expected, availability monotonically increases with the bid. However, the CDF has an extremely long tail, and there is little increase in availability after some bid threshold and only bids that fall within the steep range of the incline yield different availabilities. As the graph shows, this range of bids is quite small, providing only a narrow window where changing a bid will have a significant effect on availability. Thus, availability of spot instances is not sensitive to bidding for a large range of bid prices.

The insensitivity of bidding in determining the average cost of spot instances can similarly be seen in Figure 2(b). In this case, the cost on the y-axis is a fraction of the on-demand cost. The cost is monotonically increasing with the bid amount. However, just as with availability, the cost curve has a long tail, such that higher bids result in little or no increase in cost. This occurs because most markets always have a low and stable spot price, with the average spot price <0.2× the on-demand price. Just as with availability, bidding has little effect on the cost of spot instances, since there is no penalty for bidding high because of the second-price nature of the auction.

Finally, the frequency of revocations, as indicated by their mean-time-between-revocations (MTBR), is another important metric, since revocations incur overhead for applications that restart or migrate. Figure 2(c) shows the MTBR for different bids. The figure shows that MTBRs range from tens to hundreds of hours. In addition, the MTBRs also have a long tail in all but one market, such that bidding high does not significantly increase the MTBR and there is a wide range of bids with effectively the same MTBR. Regardless of the bid price, revocations are unavoidable when using spot instances.

In addition to the five markets discussed above, we also analyzed these properties in over 1,500 spot markets, and found that availability, cost, and MTBR are insensitive to bidding for most markets. Figure 3 is a succinct representation of our findings for the 1,500 markets. We show the length of the range of bids for which the availability, cost, and MTBR are all within 10% of the "optimal" bid. The optimal bid is the bid that yields the highest availability and MTBR for the lowest cost. In EC2, the maximum bid can be 10× the on-demand price, and thus the max bid range is 10. We see from Figure 3 that the bid range length is more than 9 for most markets, with very few outliers. This indicates that if one were to pick randomly, more than 90% of the bids would be optimal.

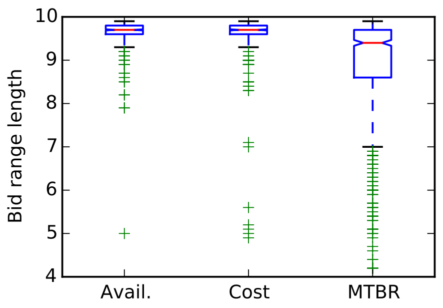

Fig. 3. *Range of bids for which availability, cost, and MTBR is within 10% of optimal across 1500 markets.*

A Simple Strategy: Based on our analysis, we argue that cloud customers need not employ sophisticated bidding and can instead use simple strategies as described next. (i) Select the spot server type carefully to reduce revocation risk. (ii) Use a bid price equal to the on-demand price. (iii) Diversify when possible by choosing multiple spot server types. (iv) If revoked, migrate the application state to a new spot server in a different market. Next, we discuss several design considerations in implementing such a strategy.

4 Mitigating Spot Instance Revocations

Applications can use the characteristics of spot markets to minimize their costs and impact of revocations. Careful spot market selection and using the appropriate fault-tolerance policies can drastically reduce the impact of revocations while also lowering costs.

Market Selection. Carefully selecting spot markets, instead of being restricted to a particular server type, can greatly increase the effectiveness of spot servers. For distributed applications, a useful strategy is to use multiple spot markets, i.e. servers in different availability zones and of different types (small, large, etc.). We have observed that price variations across markets are largely *uncorrelated* (Figure 4). In general, revocations in different markets

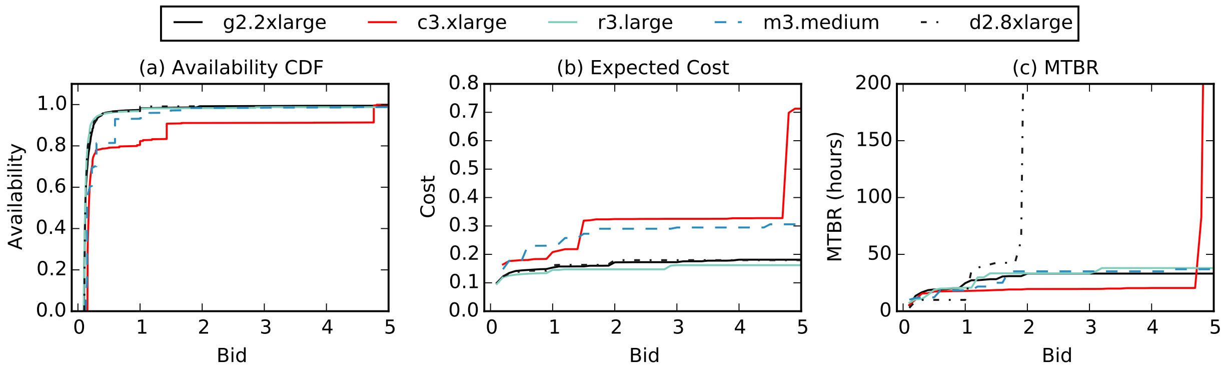

Fig. 2. *The effect of bidding on availability, expected cost, and MTBR for selected instance types. Bids and the expected costs are normalized to a factor of the corresponding on-demand price.*

do not occur at the same time. When deployed on a single market, a price spike results in revocation of *all* the servers. If instead multiple markets are used, then the application can continue to run on remaining unaffected servers.

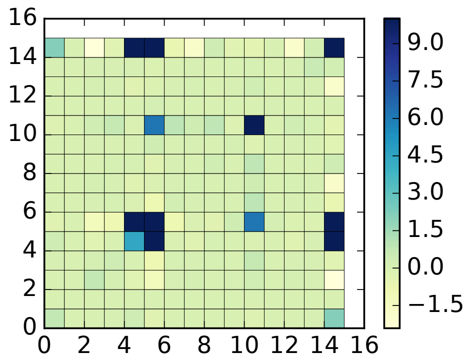

Fig. 4. *Correlation between different spot markets in the us-east-1 region. Darker squares indicate higher correlation.*

Fault Tolerance. Fault tolerance policies and migration strategies are key in light of the inevitability of revocations and the availability of multiple markets. We can treat server revocation events as fail-stop failures, and choose the suitable application-specific fault-tolerance policy. Checkpointing is a commonly used strategy, and by periodically checkpointing state to network storage, the application can resume from the most recent checkpoint. This periodic checkpoint can be performed either at the system-level using nested virtualization [6], or by using the application's built-in checkpointing mechanism (e.g., Spark [5] and MPI [4]).

Spot server revocations come with a small 120 second warning, and this warning can expand the fault tolerance choices available and reduce their overhead. For example, it may be possible for certain applications to react on revocation warning and complete a checkpoint, instead of periodically. Thus, there exist research opportunities in determining efficient checkpointing and migration strategies to exploit cheap but revocable spot servers.

Finally, we must emphasize that it is the *combination* of spot market and fault tolerance policies that determines performance and costs. An application deployed on a single market is more susceptible to failure and thus requires stronger fault tolerance, and potentially incurs a higher performance overhead. Selecting the right market may involve considering its average cost, availability and MTTR. Tools like Amazon bid-advisor [1] can help users in picking markets. A diversified *portfolio* of markets might reduce revocation risk, but at a higher cost, since it entails picking uncorrelated markets which may not have the lowest prices.

5 Conclusion and Future of Spot Markets

The analysis of historical spot price data leads us to conclude that bidding can be kept simple in today's spot markets. Instead, users should carefully select markets and fault tolerance policies for their applications.

Our results are predicated on the nature of current spot prices which are generally low but with occasional spikes. Increased usage of spot servers might change these price characteristics. If the cost and availability CDFs are no longer long-tailed, then bidding's importance will increase. However, an increased demand for spot servers might be met with an increase in supply, and the price characteristics might remain unchanged. The second-order effects of increasing spot server usage is thus unclear and remains an open question.

References

[1] Ec2 Spot Bid Advisor. https://aws.amazon.com/ec2/spot/bid-advisor/, September 2015.
[2] O. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing Amazon EC2 Spot Instance Pricing. *ACM TEC*, 1(3), September 2013.
[3] S. Higginbotham. Bidding Strategies? Arbitrage? AWS Spot Market is where Computing and Finance Meet. Gigaom, October 8th 2013.
[4] A. Marathe, R. Harris, D. Lowenthal, B. R. de Supinski, B. Rountree, and M. Schulz. Exploiting Redundancy for Cost-effective, Time-constrained Execution of HPC Applications on Amazon EC2. In *HPDC*, 2014.
[5] P. Sharma, T. Guo, X. He, D. Irwin, and P. Shenoy. Flint: Batch-Interactive Data-Intensive Processing on Transient Servers. In *EuroSys*, April 2016.
[6] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy. SpotCheck: Designing a Derivative IaaS Cloud on the Spot Market. In *EuroSys*, April 2015.
[7] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang. How to Bid the Cloud. In *SIGCOMM*, August 2015.

Prateek Sharma is a PhD student in the College of Information and Computer Sciences at the University of Massachusetts Amherst. His current research focuses on cloud computing. Contact him at prateeks@cs.umass.edu.

David Irwin is an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Massachusetts Amherst. His research interests are broadly in experimental computing systems with a particular emphasis on sustainability. Contact him at irwin@ecs.umass.edu.

Prashant Shenoy is a Professor of Computer Science at the University of Massachusetts Amherst. His current research focuses on cloud computing and green computing. He is a distinguished member of the ACM and a Fellow of the IEEE. Contact him at shenoy@cs.umass.edu.