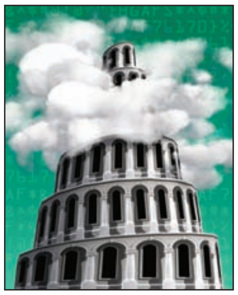

Keep It Simple: Bidding for Servers in Today's Cloud Platforms

Prateek Sharma, David Irwin, and Prashant Shenoy • *University of Massachusetts Amherst*

Dynamically priced spot servers are an increasingly popular platform on which to deploy applications. This article shows the effect of spot server bidding on application cost and availability and discusses bidding strategies and new research directions in cloud resource management and fault tolerance.

Today's infrastructure-as-a service (IaaS) cloud platforms such as Amazon Elastic Compute Cloud (EC2) and Google Cloud Platform rent computing resources on-demand in the form of virtual machine servers. Benefits of using such platforms include 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 can 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 — that is, the cloud platform can unilaterally preempt them at any time.

In the case of 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're willing to pay for a spot server, and EC2 reclaims the server whenever the server's spot price rises above the bid. Because 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 10 to 30 percent of that of on-demand servers.

Conventional wisdom has held that careful selection of bid-price is important to balance the cost–availability tradeoff — a high bid might increase costs but also increase spot server availability. Here, we show that spot instance bidding need not be complicated. We analyze empirical price data of more than 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 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 the 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.

Spot Instance Bidding

Spot instances allow cloud platforms to gain revenue from surplus idle resources. Amazon EC2 uses a market mechanism to sell this capacity

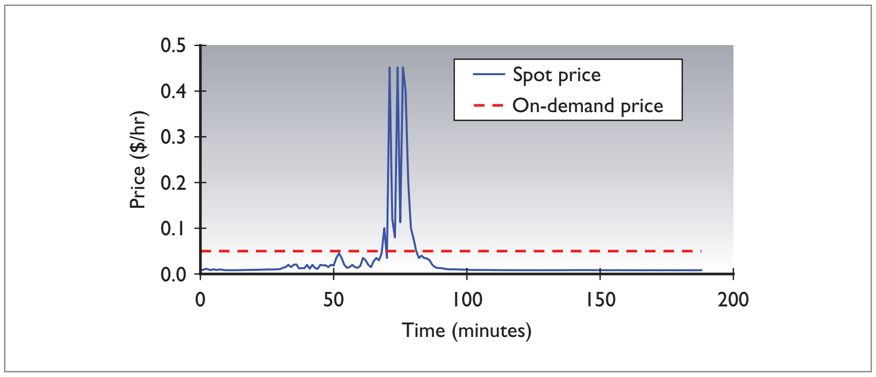

Figure 1. Variations in spot price of the m3.medium instance type. The spot price is generally much lower than the on-demand price, but shows occasional spikes.

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 multiunit second-price auction.[1] Note that the underlying supply of surplus servers in the spot pool also changes dynamically, because EC2 might 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 (see Figure 1).

To use a spot server, users place a single, fixed bid, which represents the maximum hourly price that they're willing to pay. The bids can range from zero to 10 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 might be lower than the bid. If the market price increases to more than 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 more than 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 might suffer from low availability and a higher likelihood of being revoked if the market price increases to more than the bid price. Frequent revocations might 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 might further impact application performance.

Careful selection of bids via bidding strategies has received wide attention in both research[2] and industry. Bidding strategies have been proposed for minimizing costs with different constraints (such as deadlines) for a wide range of applications (such as MapReduce, scientific computing, and so on). Bidding's complexity might be one reason why, despite its extremely low prices (70 to 90 percent less than on-demand instances), the spot market has low usage.[3] As we discuss, however, the bidding problem in today's markets (and possibly in future markets) isn't particularly important for maximizing performance and minimizing costs using spot servers.

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 — Linux instances in the region known as us-east-1.

Bidding strategies optimize the cost–availability tradeoff for spot instances: as a user increases their bid, they might pay more per hour, but their availability also increases. However, spot price data across many markets shows that a wide range of optimal bids exist 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 times those of 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 2a shows a cumulative distribution function (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 — that is, 2 is 2 times the on-demand price, and so on. As expected, availability monotonically increases with the bid. However, the CDF has an extremely long tail, and there's 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 isn't 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

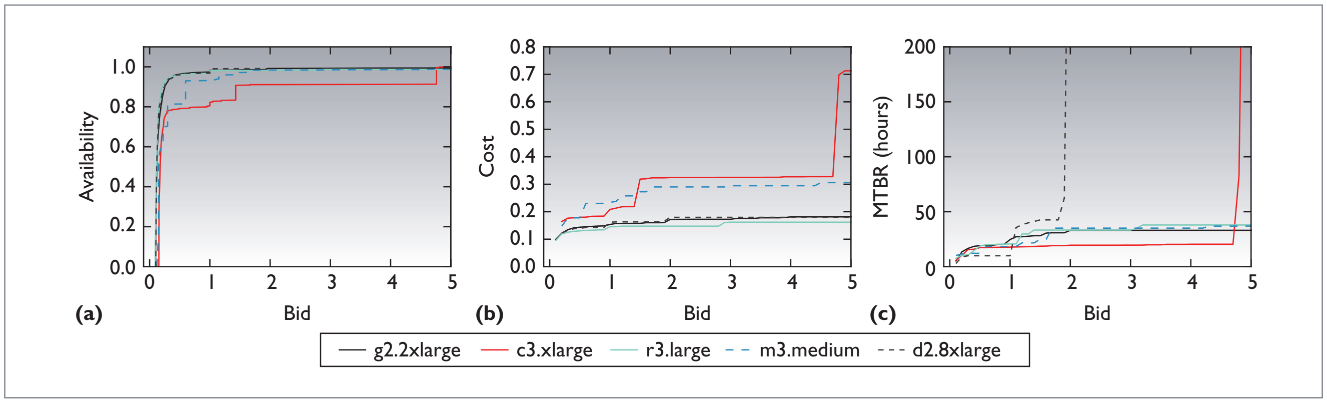

Figure 2. The effect of bidding on (a) availability, (b) expected cost, and (b) mean time between revocations (MTBR) for selected instance types. Bids and the expected costs are normalized to a factor of the corresponding on-demand price.

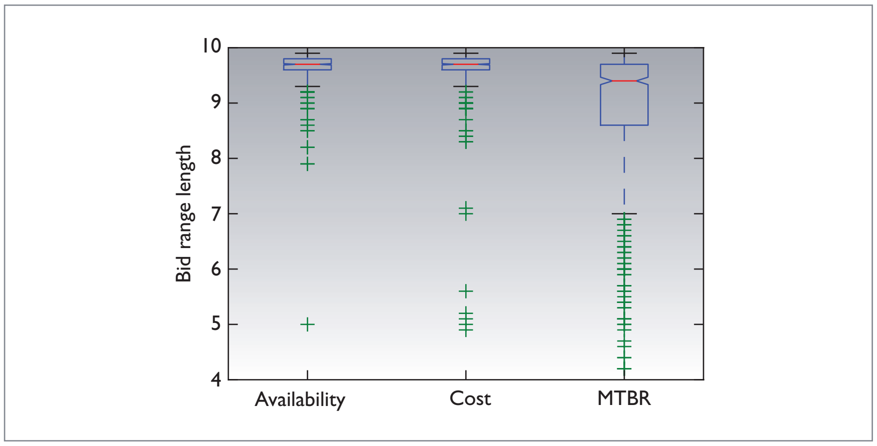

Figure 3. Range of bids for which availability, cost, and MTBR is within 10 percent of optimal across 1,500 markets.

Figure 2b. 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 times the on-demand price. Just as with availability, bidding has little effect on the cost of spot instances, because there's no penalty for bidding high due to the auction's second-price nature.

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 2c shows the MTBR for different bids. The figure shows that MTBR range from tens to hundreds of hours. In addition, the MTBR also have a long tail in all but one market, such that bidding high doesn't significantly increase the MTBR and a wide range of bids exist with effectively the same MTBR. Regardless of the bid price, revocations are unavoidable when using spot instances.

In addition to the five markets discussed previously, we also analyzed these properties in more than 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 percent 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 times the on-demand price, and thus the maximum bid range is 10. We see from Figure 3 that the bid range length is more than 9 for most markets, with few outliers. This indicates that if we were to pick randomly, more than 90 percent of the bids would be within 10 percent of the optimal.

Based on our analysis, we argue that cloud customers need not employ sophisticated bidding and can instead use simple strategies as follows. First, select the spot server type carefully to reduce revocation risk. Then use a bid price equal to the on-demand price. Diversify when possible by choosing multiple spot server types. And finally, 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.

Mitigating Spot Instance Revocations

Applications can use the characteristics of spot markets to minimize their costs and the 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 — that is, servers in different availability zones and of different types (small, large, and so on). We observed that price variations across markets are largely uncorrelated (see Figure 4). In general, revocations in different markets don't 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.

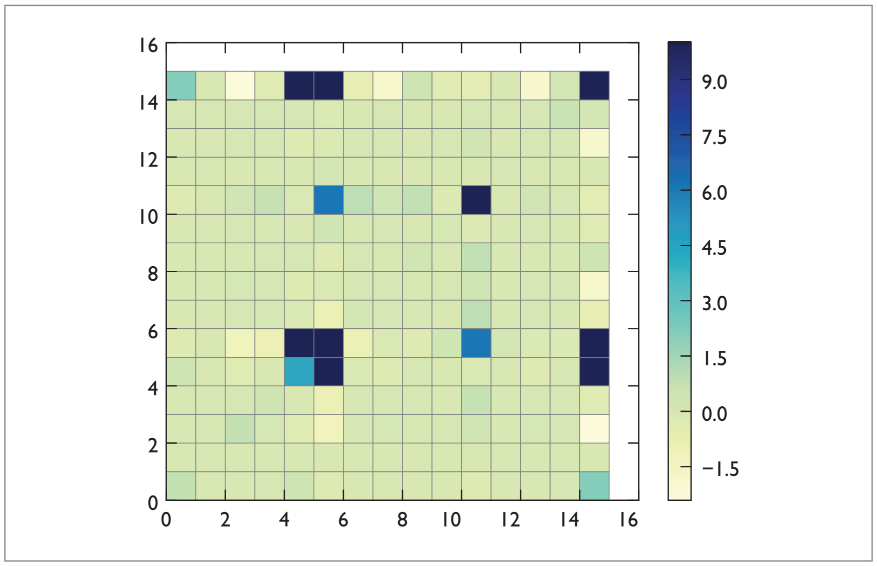

Figure 4. Correlation between different spot markets in the us-east-1 region. Darker squares indicate higher correlation.

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,[4] or by using the application's built-in checkpointing mechanism.[5,6]

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 might be possible for certain applications to react on revocation warning and complete a checkpoint, instead of periodically checkpointing. Thus, there exist research opportunities in determining efficient checkpointing and migration strategies to exploit inexpensive but revocable spot servers.

Finally, we must emphasize that it's 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 might involve considering its average cost, availability, and MTBR. Tools such as Amazon Spot Bid Advisor (see aws.amazon.com/ec2/spot/bid-advisor) can help users in picking markets. A diversified portfolio of markets could reduce revocation risk, but at a higher cost, because this entails picking uncorrelated markets, which might not have the lowest prices.

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 are thus unclear and remain an open question.

References

1. O. Ben-Yehuda et al., "Deconstructing Amazon EC2 Spot Instance Pricing," *ACM Trans. Economics and Computation* (TEC), vol. 1, no. 3, 2013, article no. 16.
2. L. Zheng et al., "How to Bid the Cloud," *Proc. ACM Conf. Special Interest Group on Data Comm.*, 2015, pp. 71–84.
3. S. Higginbotham, "Bidding Strategies? Arbitrage? AWS Spot Market Is Where Computing and Finance Meet," *Gigaom*, 8 Oct. 2013; https://gigaom.com/2013/10/08/bidding-strategies-arbitrage-aws-spot-market-is-where-computing-and-finance-meet.
4. P. Sharma et al., "SpotCheck: Designing a Derivative IaaS Cloud on the Spot Market," *Proc. 10th European Conf. Computer Systems*, 2015; doi:10.1145/2741948.2741953.

5. P. Sharma et al., “Flint: Batch-Interactive Data-Intensive Processing on Transient Servers, *Proc. 11th European Conf. Computer Systems*, 2016; doi:10.1145/2901318.2901319.
6. A. Marathe et al., “Exploiting Redundancy for Cost-effective, Time-Constrained Execution of HPC Applications on Amazon EC2,” *Proc. 23rd ACM Symp. High-Performance Parallel Distributed Computing*, 2014; doi:10.1145/2600212.2600226.

Prateek Sharma is a PhD student in the College of Information and Computer Sciences at the University of Massachusetts Amherst. His research interests focus on cloud computing. Sharma has an MS in computer science from IIT-Bombay. 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 include experimental computing systems with a particular emphasis on sustainability. Irwin has a PhD in computer science from Duke University. Contact him at irwin@ecs.umass.edu.

Prashant Shenoy is a professor of computer science at the University of Massachusetts Amherst. His research interests include cloud computing and green computing. Shenoy has a PhD in computer science from the University of Texas at Austin. He’s an IEEE Fellow and ACM distinguished member. Contact him at shenoy@cs.umass.edu.