Cloud Spot Markets are Not Sustainable: The Case for Transient Guarantees

Supreeth Subramanya, Amr Rizk, and David Irwin
University of Massachusetts Amherst

Abstract

Computational spot markets enable users to bid on servers, and then continuously allocates them to the highest bidder: if a user is "out bid" for a server, the market revokes it and re-allocates it to the new highest bidder. Spot markets are common when trading commodities to balance real-time supply and demand—cloud platforms use them to sell their idle capacity, which varies over time. However, server-time differs from other commodities in that it is "stateful": losing a spot server incurs an overhead that decreases the useful work it performs. Thus, variations in the spot price actually affect the inherent value of server-time bought in the spot market. As the spot market matures, we argue that price volatility will significantly decrease the value of spot servers. Thus, somewhat counter-intuitively, spot markets may not maximize the value of idle server capacity. To address the problem, we propose a more sustainable alternative that offers a variable amount of idle capacity to users for a fixed price, but with *transient guarantees*

1 Introduction

Infrastructure-as-a-Service (IaaS) platforms, such as Amazon's Elastic Compute Cloud (EC2) and Google Compute Engine (GCE), are growing rapidly as users migrate to the cloud. While these platforms leverage statistical multiplexing at massive scales to reduce costs, they still experience daily and seasonal fluctuations in demand. Thus, to prevent rejecting server requests, platforms must provision for their expected peak demand, which often results in a large number of idle servers.

Maintaining idle servers is a waste of money: it wastes the capital expenses used to purchase and house the servers and the operational expenses used to power and cool them. Thus, to gain additional revenue and flexibility, platforms are increasingly renting out idle servers to users, while reserving the right to reclaim them to service higher-priority requests. EC2 pioneered this approach with its spot market, which enables users to place a bid for one or more servers. If the bid price is greater than the servers' current *spot price*, EC2 allocates them to the user, who pays the per-hour spot price for them. However, if the spot price, which varies in real time, ever rises above the user's bid price, EC2 revokes the servers. Spot markets are commonly used when trading commodities to balance real-time supply and demand. Commodities sold in a spot market are delivered immediately, which differs from a *futures market* where commodities are delivered at a future time. Spot markets enable users that purchase commodities in a futures market (based on what they expect their future demand will be) to resolve in real time the difference between their expected demand and their actual demand by buying or selling resources.

Cloud platforms increasingly resemble a commodities market for server-time. For example, in addition to the spot market, EC2 also manages a reserved market where users may sell the remaining term on server reservations they have previously purchased. The reserved market is effectively a futures market for server-time. While market-based resource allocation of server-time has been a research topic for nearly 50 years [12], EC2 is the first system to employ market-based allocation at scale. As prior work describes, market-based allocation is attractive, since it determines the "right" price to balance supply and demand [10]: it automatically prioritizes access to limited resources (by giving them to the users willing to pay the highest price) and eliminates idle resources (by lowering the price until users are willing to buy them).

However, spot markets are inherently volatile, as prices vary due to changes in supply and demand. Mature markets should follow the efficient market hypothesis, which states that you cannot "beat the market" by predicting future prices, as the current price already reflects all available information. Thus, mature markets are also inherently unpredictable. Prior research has not considered how volatile and unpredictable markets affect application performance. As we discuss, the volatility and predictability of spot market prices directly affect the value of the underlying servers. Server-time fundamentally differs from other commodities in that it is "stateful": losing a server incurs an overhead that decreases the "useful" work it performs. Thus, while high market volatility in the five-minute energy spot market does not decrease the amount of energy purchased, *it does reduce the amount of useful server-time purchased.*

Given this relationship between useful server-time and market dynamics, we argue that, as computational spot markets mature, the value of the resources they allocate will decrease. This is currently not a problem in EC2 because its spot market is highly under-utilized. As a result, the spot price is low and relatively stable across most of EC2's roughly 4500 spot markets. For example, in many markets the spot price of a server is $\sim 10\times$ less than the price of the equivalent on-demand server (which EC2 cannot revoke). In addition, third parties estimate that only 3-5% of the servers allocated by EC2 currently come from the spot market [3]. However, prior research and many startups [5, 7] propose to exploit the existing arbitrage opportunities between the price of on-demand and spot servers. This body of work proposes to dynamically shift computation between spot and on-demand servers as spot prices change to minimize cost [9, 11].

As users (and applications) become more sophisticated, we expect them to increasingly exploit these arbitrage opportunities to lower their costs. Unfortunately, the more these arbitrage opportunities are exploited, the higher, more volatile, and more unpredictable spot prices will become. This will, in turn, decrease the inherent value of a spot server—in terms of the amount of useful work it can perform—due to overhead from i) recomputing lost state after a spot server revocation or ii) migrating or saving state in anticipation of a spot server revocation. Of course, assuming rational users, the market should reach an equilibrium price that balances expectations of price volatility with the value of server-time at that volatility level. However, this equilibrium price may be significantly less than the value (and price) a cloud platform could offer using other pricing mechanisms.

In this paper, we propose a more sustainable alternative to spot markets based on offering users *transient guarantees* when selling idle capacity. The goal of transient guarantees is to allow platforms to retain the freedom to reclaim idle capacity when necessary, but provide users with statistical assurances about its availability, volatility, and predictability. Unlike spot markets, these assurances enable users (and platforms) to maximize the value of such *transient servers*. For example, a platform might sell idle capacity for a fixed price, similar to GCE's Preemptible Instances, but provide a transient guarantee on its mean-time-to-revocation.

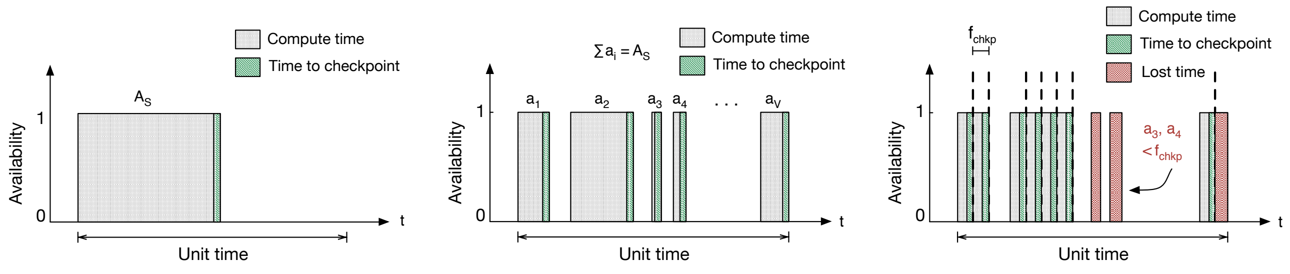
2 Example Application

To understand how market dynamics affect server value, consider stateful batch applications that perform significant computations on in-memory datasets and only periodically checkpoint their intermediate state to disk. Examples of such applications include scientific simulations, e.g., for weather forecasting and drug discovery, and next generation big data frameworks, such as Spark [14] and Naiad [6], that operate on distributed volatile in-memory state without writing it to disk. The loss of in-memory state due to a failure or, equivalently, a spot server revocation requires these applications to restart from their last checkpoint. Importantly, the overhead of recomputation and checkpointing reduce the amount of server-time devoted to useful computation.

While most prior work focuses on the *availability* of spot servers, the value derived from spot servers for such stateful applications is actually a function of price *volatility* and *predictability*. The frequency and predictability of revocations determine how applications should tune fault-tolerance mechanisms to minimize the impact of revocations. For the stateful applications above, this translates to setting the optimal checkpoint frequency that minimizes running time by balancing recomputation and checkpointing overhead. Other applications might use other fault-tolerance mechanisms, e.g., consensus protocols, primary-backup systems, erasure codes, etc. Each of these mechanisms incur overhead that is related to the availability, volatility, and predictability of spot servers, which reduces the server-time applications are able to devote to useful computation.

To illustrate, for a simple single-node batch job, the optimal checkpointing interval that minimizes job running time (when accounting for the overhead of recomputation and checkpointing) is $t_{opt} \sim \sqrt{2 * \delta * MTTR}$, where δ is the time to write each checkpoint and MTTR is the mean-time-to-revocation [2, 4, 8, 13]. Thus, every t_{opt} interval, the application must spend δ time writing a checkpoint. Since a spot server's MTTR is a function of price volatility, the more volatile the prices, the less useful work on average the job can perform per unit time. Based on the equation above, given an accurate MTTR, users can set their checkpointing interval to minimize overhead and running time. Our premise is that, as the market matures to follow the efficient market hypothesis, the spot price will become more volatile and less predictable, which will decrease the MTTR and the amount of useful computation extracted from a spot server.

Based on the relationships above, we can derive the inherent value of a spot server for our batch job in terms of volatility, checkpointing/recomputation overhead, and the price of an equivalent on-demand server. In this case, the expected completion time $E[T_j]$ for a job j



(a) Available, Not Volatile, Predictable (b) Available, Volatile, Predictable (c) Available, Volatile, Unpredictable

Figure 1: Availability, volatility, and predictability are three distinct metrics that affect spot server value.

with running time T_j on a spot server will be $E[T_j] = T_j + \frac{T_j}{t_{opt}}\delta + \frac{T_j}{MTTR} * \frac{t_{opt}}{2}$, where the first term is the job's actual running time, the second term is the overhead from checkpointing (at the optimal frequency), and the last term is the expected recomputation overhead across all revocations. Based on this analysis, the spot server is only worth $\frac{T_j}{E[T_j]}$ of the on-demand server. That is, if an on-demand server costs p_o, then a user should not be willing pay more than $p_e = p_o * \frac{T_j}{E[T_j]}$ for the equivalent spot server. However, if the current spot price $p_c < p_e$, an arbitrage opportunity exists that rational users have an incentive to exploit. We call p_e the *equilibrium price* of the spot server—it represents its maximum value as a function of the price of an on-demand server. Note that this equilibrium price is application-dependent, as other types of applications might value spot servers differently. For example, the value of spot servers for stateless applications, i.e., those that only serve static content or write all intermediate state to disk, is largely a function of availability (and not volatility or predictability).

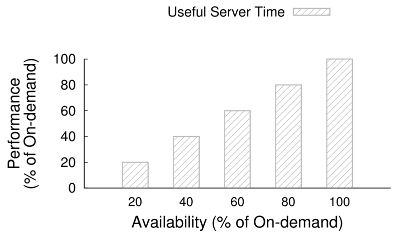
Consider a specific example where a batch job takes 12 hours to complete uninterrupted on an on-demand server. However, on a spot server, after considering the time δ to write each checkpoint and the MTTR (at a certain bid price), the job takes 24 hours to finish. Also assume that a spot server is always available in one of EC2's 4500 spot markets, such that upon revocation the job resumes immediately on another available spot server. Thus, since the job takes twice as long running on spot servers, the performance is 50% that of the on-demand server. As a result, the maximum a user should be willing to pay for spot server is 50% of the on-demand price. Put another way, if a spot server's price is >50% of the on-demand price, the overall cost of executing the job on spot servers will actually be more than on an on-demand server due to the overhead of recomputation and checkpointing.

Note that EC2 only advertises the absolute difference in price between spot and on-demand servers, e.g., spot servers cost 50-90% less than equivalent on-demand servers. However, as our simple example above illustrates, spot servers are worth fundamentally less than on-demand servers. The actual discount a spot server provides must also consider its performance relative to an on-demand server. In the example above, if the spot price is 50% of the on-demand price, then spot servers offer no real discount over on-demand servers. Finally, our analysis above assumes the MTTR is well-known. However, setting an incorrect MTTR will further decrease the useful server-time. Of course, accurately estimating the MTTR requires accurately predicting future spot prices, which will become more difficult as the market matures.

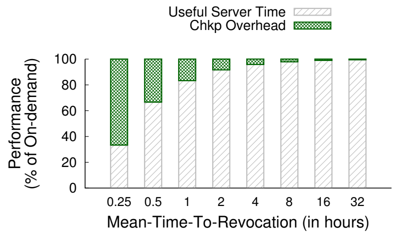
3 Characterizing Spot Server Value

The previous section highlights three key metrics for characterizing spot servers: availability, volatility, and predictability. Availability is the percentage of time a spot server is available; in EC2, this translates to the percentage of time the spot price is below a user's bid price. By contrast, volatility is the frequency of revocations a spot server experiences; in EC2, this translates to the frequency at which the spot price rises from below to above a user's bid price. Finally, predictability captures the stationarity of the spot price time-series; in EC2, predictability is a measure of how much the mean and variance of the spot price change over time.

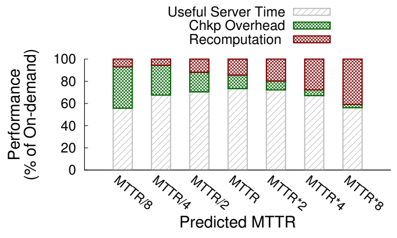
Figure 1 illustrates, in the context of our simple batch job, that these three metrics are distinct from each other. Figure 1(a) shows a time-series of spot server availability that is not volatile and highly predictable. In this case, there is only a single revocation, and since the revocation is predictable, the application need only checkpoint immediately before the revocation occurs, thereby minimizing its overhead (in green) and maximizing the useful work it performs (in grey). Figure 1(b) shows a similar time-series with the same availability over time, but with high volatility that includes many revocations. In this scenario, the application incurs more overhead (in green) than before because it needs to checkpoint more frequently, but, since the revocations are highly predictable, it still need only checkpoint immediately prior to a revocation. Finally, Figure 1(c) shows a time-series with the



(a) Availability



(b) Volatility



(c) Predictability

Figure 2: Impact on performance of a spot instance when (a) varying availability, (b) varying volatility at a given level of availability (c) varying predictability at a given level of availability and volatility.

same availability but with high volatility and low predictability. Here, the application incurs a higher checkpointing overhead (in green), as it does not know precisely when to checkpoint, and also incurs recomputation overhead (in red) when a revocation occurs unexpectedly.

Similarly, Figure 2 shows how the value of spot servers vary, measured as a percentage of their performance relative to on-demand servers, for each metric. Figure 2(a) simply shows that the percentage of time a spot server is available is linearly related to its rate of computation: if spot server is only available 50% of the time, its rate of computation is at most 50% that of on-demand. However, the availability of any single spot server is not a particularly significant metric, as EC2's spot market is so large that spot servers of some type are always available (for a certain price). Thus, rather than waiting for a specific spot server price to drop before resuming an application, rational users should immediately migrate to the next-lowest priced spot server and continue execution. Since some spot (or on-demand) server is always available, volatility and predictability are much more important in assessing the value of spot servers.

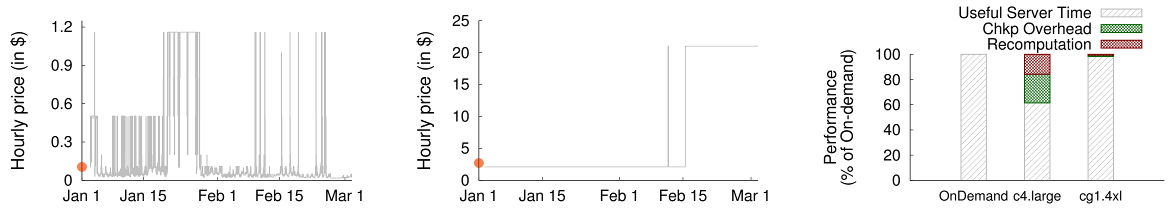
Figure 2(b) shows that the more volatile a spot server, i.e., as its MTTR decreases, the less valuable it is relative to an on-demand server. In this case, to isolate volatility from availability, we assume spot servers are always available, but are revoked and lose state according to the MTTR. In these graphs, the equilibrium price can be computed by simply multiplying the on-demand price by the percentage of useful server-time on the y-axis. The graph shows that if the MTTR were to decrease to between 15 and 30 minutes, the actual amount of server-time provided by a spot server would only be 30%-70% that of an equivalent on-demand server. This would effectively eliminate the actual discount that spot servers provide; that is, they might cost 30% of an on-demand server but they would only provide 30% of the performance. Similarly, Figure 2(c) shows that as spot server revocations become less predictable, their value relative to an on-demand server further decreases. Here, the *x*-axis is the difference between an application's predicted MTTR and the actual MTTR (for a fixed volatility of MTTR equal to 4 hours and 100% availability).

Our results show that a low availability, high volatility, and low predictability can significantly decrease the value of spot servers in terms of their performance relative to on-demand servers. Since each characteristic is a function of changes in the spot price, the value of the spot servers is a function of the magnitude, variance, and stationarity of the spot price. The decrease in value due to spot market volatility is currently not an issue in EC2, as its spot market is highly under-utilized. However, if market volume increases and the spot market becomes more highly utilized, the market is likely to become more volatile and less predictable, thereby decreasing the value of the resources bought in it. There are already indications that this might happen.

Figures 3(a) and (b) shows spot prices for the `c4-large` and `cg1-4xlarge` over the past two months. We expect that the c4-large is a much higher volume and mature market than the cg1-4xlarge market, since the c4-large is a current generation compute-oriented server while the cg1-4xlarge is a more exotic GPU-oriented server that is deprecated. The graphs show that prices for the more mature c4-large market are much more volatile. Figure 3(c) then plots the performance of these spot servers for our example batch job as a function of the equivalent on-demand server (given a bid equal to the on-demand price). The figure shows that the value of the spot server in the more stable cg1-4xlarge market is close to that of an on-demand server, while the value of the spot server in the volatile c4-large market is only 60% that of the on-demand server. In this case, since the average spot price of $0.212 is already more than double the on-demand price, using an unreliable c4-large spot server is a 250% cost increase over a reliable on-demand server, when considering its revocation overhead.

4 Transient Guarantees

Since the value of spot servers will decrease substantially as the market matures, the spot market may not maximize the value of a variable supply of idle capacity. Of course, there are other ways to sell transient servers. For ex-



(a) c4-large (b) cg1-4xlarge (c) Performance

Figure 3: Spot prices in more mature markets (a) are much more volatile and less predictable than spot prices in less mature markets (b). As a result, server-time in more mature markets is worth less than in less mature markets (c).

ample, GCE recently introduced Preemptible Instances, which it sells for a fixed per-hour price but may revoke at any time (and always within 24 hours). However, one problem with GCE's approach is that it conveys no information to the user about revocation characteristics, e.g., the availability, volatility, and predictability. Thus, it is impossible for users to quantify the true value (or equilibrium price) of Preemptible Instances. In contrast, since EC2 releases spot price history, users can at least estimate the value of spot instances based on historical data.

To address the problem, we propose a new abstraction called a *transient guarantee*, which provides users statistical assurances about the availability, volatility, and predictability of transient servers. EC2 is already experimenting with one type of transient guarantee in the form of spot blocks, which provides a spot server for a fixed block of time. Spot blocks make the revocation time entirely predictable, as EC2 guarantees to revoke the server at the end of the time block. Based on our analysis in the previous section, spot blocks are worth much more than spot servers, since applications only need to incur the overhead of checkpointing once (at the end of the time block), and not at a fixed frequency (as is the case when revocations are unpredictable). Of course, spot blocks require a platform to precisely predict its variations in idle capacity, which is likely not possible at large scales as it would require platforms to know *exactly* when customers would make new requests for on-demand resources. EC2 likely only offers a small fraction of its idle capacity as spot blocks to prevent rejecting requests for on-demand instances because it cannot revoke a spot block server.

Platforms may be able to more accurately predict the statistical characteristics of the supply of idle capacity over time, e.g., its mean and variance. A platform could also exploit this information by offering a transient guarantee that ensures a specific MTTR for transient servers, enabling applications to correctly tune fault-tolerance mechanisms and value these servers. Predictions of the statistical characteristics of idle capacity by the platform are likely to be much more accurate than predictions of spot prices by users, as the former are only a function of supply while the latter are a function of both supply and demand, e.g., the number of bids by users and their value. In addition, unlike spot blocks, statistical guarantees on availability, volatility, or predictability allow platforms to retain some flexibility to revoke servers when necessary, while enabling users to maximize server value.

Of course, platforms may wish to relax an MTTR, e.g., by guaranteeing a shorter MTTR than predicted, to account for inaccurate supply predictions. Transient guarantees expose the relationship between future knowledge of supply and value: the more accurate a platform can predict their supply, the higher the value of the transient guarantee they can offer and the more revenue they can generate from their resources. Recent work exploits this property to define an economy class of on-demand servers using a fraction of the idle capacity (although without considering overheads), which has 98% availability and costs much less than on-demand servers [1]. Transient guarantees generalize this property, and apply it to any characteristic that affects spot server value.

Transient guarantees raise other interesting research problems. For example, how do users verify transient guarantees that are based on statistical properties? While large-scale users can average their performance across a large number of requests, small-scale users will not necessarily know if the platform is maintaining transient guarantees across its user base. We are currently exploring crowd-sourced techniques for verifying transient guarantees that enable small users to anonymously pool their revocation data to verify these guarantees.

5 Conclusion

Based on the relationship between market dynamics and spot server value, we argue that spot markets are not sustainable. Instead, we propose a new abstraction of a transient guarantee, which enables platforms to retain the freedom to revoke servers while maximizing their value. In maximizing value, platforms can either increase the price of transient servers (to near the equilibrium price) or provide better performance for users than competitors at the same price. We are currently exploring different types of transient guarantees, and their relative benefit compared to volatile and unpredictable markets.

Acknowledgements. Work supported by the NSF (#1422245) and a Google Faculty Research Award.

References

[1] Carvalho, M., Cirne, W., Brasileiro, F., and Wilkes, J. Long-term SLOs for Reclaimed Cloud Computing Resources. In *SoCC* (November 2014).

[2] Daly, J. A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps. In *Future Generation Computer Systems* (2006), vol. 22.

[3] Higginbotham, S. Bidding Strategies? Arbitrage? AWS Spot Market is where Computing and Finance Meet. Gigaom, October 8th 2013.

[4] Khatua, S., and Mukherjee, N. Application-centric Resource Provisioning for Amazon EC2 Spot Instances. In *EuroPar* (August 2013).

[5] Lardinois, F. Spotinst, which helps you buy AWS spot instances, raises $2m Series A. TechCrunch, March 8th 2016.

[6] Murray, D., McSherry, F., Isaacs, R., Isard, M., Barham, P., and Abadi, M. Naiad: A Timely Dataflow System. In *SOSP* (October 2013).

[7] Novet, J. Amazon pays $20M-$50M for ClusterK, the startup that can run apps on AWS at 10% of the regular price. VentureBeat, April 29th 2015.

[8] Sharma, P., Guo, T., He, X., Irwin, D., and Shenoy, P. Flint: Batch-Interactive Data-Intensive Processing on Transient Servers. In *EuroSys* (April 2016).

[9] Sharma, P., Lee, S., Guo, T., , Irwin, D., and Shenoy, P. SpotCheck: Designing a Derivative Cloud on the Spot Market. In *Eurosys* (April 2015).

[10] Shneidman, J., Ng, C., Parkes, D., AuYoung, A., Snoeren, A., Vahdat, A., and Chun, B. Why Markets Could (but don't currently) Solve Resource Allocation Problems in Systems. In *HotOS* (June 2005).

[11] Subramanya, S., Guo, T., Sharma, P., Irwin, D., and Shenoy, P. SpotOn: A Batch Computing Service for the Spot Market. In *SoCC* (August 2015).

[12] Sutherland, I. A Futures Market in Computer Time. *CACM 11*, 6 (1968).

[13] Voorsluys, W., and Buyya, R. Reliable Provisioning of Spot Instances for Compute-Intensive Applications. In *AINA* (2012).

[14] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M., Shenker, S., and Stoica, I. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *NSDI* (April 2012).