The Price is (Not) Right: Reflections on Pricing for Transient Cloud Servers

David Irwin, Prashant Shenoy, Pradeep Ambati, Prateek Sharma[†], Supreeth Shastri[‡], Ahmed Ali-Eldin
University of Massachusetts Amherst [†]Indiana University [‡]University of Texas at Austin

***Abstract*—Amazon introduced *spot instances* in December 2009, enabling "customers to bid on unused Amazon EC2 capacity and run those instances for as long as their bid exceeds the current Spot Price." Amazon's real-time computational spot market was novel in multiple respects. For example, it was the first (and to date only) large-scale public implementation of market-based resource allocation based on dynamic pricing after decades of research, and it provided users with useful information, control knobs, and options for optimizing the cost of running cloud applications. Spot instances also introduced the concept of *transient cloud servers* derived from variable idle capacity that cloud platforms could revoke at any time. Transient servers have since become central to efficient resource management of modern clusters and clouds. As a result, Amazon's spot market was the motivation for substantial research over the past decade.**

Yet, in November 2017, Amazon effectively ended its real-time spot market by announcing that users no longer needed to place bids and that spot prices will "...adjust more gradually, based on longer-term trends in supply and demand." The changes made spot instances more similar to the fixed-price transient servers offered by other cloud platforms. Unfortunately, while these changes made spot instances less complex, they eliminated many benefits to sophisticated users in optimizing their applications. This paper provides a retrospective on Amazon's real-time spot market, including its advantages and disadvantages for allocating transient servers compared to current fixed-price approaches. We also discuss some fundamental problems with Amazon's spot market, which we identified in prior work (from 2016), that predicted its eventual end. We then discuss potential options for allocating transient servers that combine the advantages of Amazon's real-time spot market, while also addressing the problems that likely led to its elimination.

.

I. Introduction

Amazon's Elastic Compute Cloud (EC2) introduced *spot instances* [1] on December 14th, 2009 [2], enabling "...customers to bid on unused Amazon EC2 capacity and run those instances for as long as their bid exceeds the current Spot Price." Amazon's real-time computational spot market was novel in multiple respects: it was the first (and to date only) large-scale public implementation of market-based allocation of computing resources based on dynamic pricing after decades of research [3], [4], [5], and it provided users with useful information, control knobs, and options for optimizing the cost of running cloud applications. Perhaps most importantly, spot instances also first introduced the concept of *transient servers* [6], [7], which are derived from variable idle capacity, and that platforms may revoke at any time. Transient servers have since become central to efficient resource management in modern clusters and clouds [8], [9], [10].

Spot instances were (and still are) highly attractive to users due to their low spot price, which is stems from their low reliability and is typically 50-90% lower than the fixed price of *on-demand servers*. As a result, EC2's spot market was the motivation for substantial research in multiple areas over the past decade, e.g., on optimizing bidding strategies [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], gracefully handling revocations [7], [21], [22], [23], pricing in risk when selecting cloud servers [24], [25], etc.

Yet, in November 2017, Amazon effectively ended its real-time computational spot market by announcing that users no longer needed to place bids and that spot prices will "...adjust more gradually, based on longer-term trends in supply and demand [26]." In addition, EC2 replaced the "bid price" with a "maximum price," and made setting it optional, such that the default "maximum price" is equal to the instance type's corresponding on-demand price [27]. The changes made spot instances more similar to the fixed-price transient servers offered by other cloud platforms. Unfortunately, while these changes made spot instances less complex, they eliminated many benefits to sophisticated users in optimizing their applications. The goal of this paper is to provide a retrospective on Amazon's real-time computational spot market, including its advantages and disadvantages for allocating idle cloud server capacity compared to current fixed-price approaches by Google and Microsoft's cloud platforms. We first provide a brief overview of spot instances and their intended purpose.

Spot instances were originally designed for applications that are tolerant to delays and performance degradation due to periodic resource unavailability, and thus can make use of EC2's fluctuating amount of idle capacity as it becomes available. EC2's (and other cloud provider's) idle capacity is likely large on average, as they must provision their total server capacity for expected peak loads that rarely occur, or risk annoying their users by rejecting too many of their VM requests. Thus, spot instances enabled EC2 to earn additional revenue from their (otherwise idle) server capacity, which is a sunk cost. Importantly, since its idle capacity fluctuates, EC2 may need to *revoke* (or shutdown) spot instances at any time to satisfy new requests for on-demand (or reserved instances), which effectively reduces the supply of idle capacity. In contrast, EC2 does not forcibly revoke on-demand (or reserved) instances. Figure 1 depicts this relationship between reserved, on-demand, and spot instance pools, where the spot pool fluctuates dynamically based on the unused reserved and on-demand instances. Anytime EC2 allocates a new reserved or

978-1-7281-1856-7/19/$31.00 ©2019 IEEE

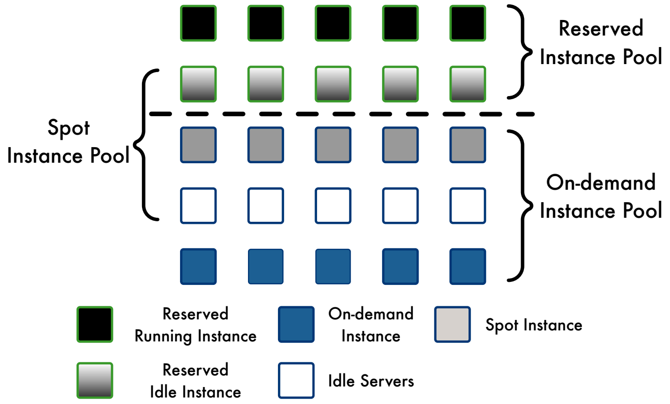


Fig. 1. Relationship between reserved, on-demand, spot instance pools hosted on same set of physical cloud servers.

on-demand request, it reduces the resources available in the spot instance pool and may result in a revocation.

Due to the possibility of revocation and their lack of availability guarantees, spot instances are much less valuable than on-demand or reserved instances, which is typically reflected in a spot price that is 2-10× less than the on-demand price. In this case, by "value" we mean the useful performance derived from a spot instance after accounting for the overhead of revocation, which may include re-executing work lost on revocation or implementing fault-tolerance mechanisms, such as checkpointing and replication, to mitigate the impact of revocation [28], [29]. Spot instances' low cost is especially attractive if applications do not require availability guarantees and can gracefully handle periodic server revocations.

Of course, the key defining characteristic of spot instances from the outset was their dynamic spot price that varied over time. While EC2's documentation never explicitly stated the pricing algorithm, it did say that the spot price "...fluctuate[d] periodically depending on the supply and demand of Spot instance capacity," and strongly implied that the price was set equal to the lowest winning bid in a continuous sealed-bid multi-unit uniform price auction. EC2's global spot market was (and is) massive—encompassing thousands of instance types with distinct spot price dynamics—since it operated a different spot market for each type of VM in each Availability Zone (AZ) of each geographic region. EC2 also publishes spot prices in real-time and makes the previous 3 months of data accessible for download. As we discuss, the real-time spot price data provided important visibility into EC2's spot market and the demand for different instance types, as well as the underlying load dynamics of EC2's cloud data centers.

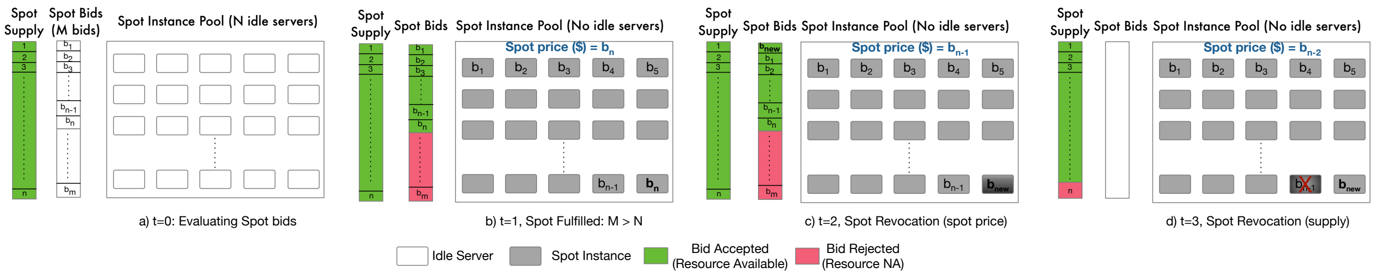
Analyses of spot price data showed that EC2 periodically changed the pricing algorithm, and suggested it may have sometimes deviated from a simple uniform price auction, e.g., by having a hidden reserve price [30], [31]. Importantly, we note that, since EC2 was the sole supplier, spot instances were never sold in a "real" market, i.e., where the spot price is determined by matching supply and demand among competing buyers and sellers, as EC2 could manipulate the price by manipulating the supply. For example, EC2 could reduce the resources available in the spot pool even if there were no reserved or on-demand requests, which would drive up the real-time spot price. That said, EC2 did ensure that spot instances would run "...as long as their bid exceeds the current Spot Price." Importantly, *enforcing this revocation policy* required *EC2 to set the instantaneous real-time spot price such that it matched the instantaneous supply (of idle server capacity) and demand (set of user bids and values).*

To understand why, consider the scenario in Figure 2(a) with a spot price equal to the N^{th} highest bid for N available spot instances (such that each user bids for a single instance). In this case, the N available spot instances are allocated to the N highest bids (b). However, if a new user bids greater than the highest current bid (b_{new} in (c)) and the price reflects the instantaneous matching of supply and demand, then the spot price should change to the $(N-1)^{th}$ highest bid, causing the user with the N^{th} highest bid to be revoked.

The revoked user with the N^{th} highest bid should observe that the spot price has risen above their bid price, which is consistent with the revocation policy. The new user should be allocated the revoked instance, since their bid is greater than the spot price. Similarly, if the supply of idle server capacity decreases (d), the spot price also increases such that any users of revoked spot instances observes that the spot price rises above their bid price. In contrast, if EC2 *did not* change the spot price in these cases, then spot instances would have to be revoked without the spot price rising above the corresponding bid price, which contradicts the revocation policy. Thus, even though EC2's spot market was not a "real market," its original revocation policy did necessitate that it faithfully alter the spot price to reflect the instantaneous supply and demand, so that all users observed a consistent revocation policy. This resulted in periods of high spot price volatility characterized by sharp price spikes and dips, as noted in prior work [24], [25], [32], [33]. This volatility is undesirable from a user's perspective.

Undesirable spot price volatility likely led to Amazon's November 2017 announcement that spot prices will "...adjust more gradually, based on longer-term trends in supply and demand." However, since these more gradual price adjustments likely do not derive from instantaneously matching supply and demand, then EC2 cannot enforce their original revocation policy above. That is, *users may have spot instances revoked, or may not be able to acquire spot instances, when their "maximum price" is above the spot price*. There is evidence of such behavior in the EC2's spot market after the change [34].

As we discuss, this has profound effects on optimizing applications for spot instances. The gradual spot price changes and decoupling of spot price from revocations make EC2 spot instances similar to transient server offerings from Google Cloud Platform (GCE) (called Preemptible VMs [35]) and Microsoft Azure (called Low-priority Batch VMs). In both cases, GCE and Azure simply charge a fixed price for transient servers and reserve the right to revoke them at any time. To illustrate, Figure 3 shows the spot price of a `m4.4xlarge` over 3 months both before and after the announcement above; the horizontal line in the figure represents the on-demand


Spot Supply
Spot Bids (M bids)
Spot Instance Pool (N idle servers)
Spot Bids
Spot Instance Pool (No idle servers)
Spot price ($) = b_n
Spot price ($) = b_{n-1}
Spot price ($) = b_{n-2}
b_1
b_2
b_3
b_4
b_5
b_{n-1}
b_n
b_m
b_{new}
1
2
3
n
Idle Server
Spot Instance
Bid Accepted (Resource Available)
Bid Rejected (Resource NA)

a) t=0: Evaluating Spot bids

b) t=1, Spot Fulfilled: M > N

c) t=2, Spot Revocation (spot price)

d) t=3, Spot Revocation (supply)

Fig. 2. Scenario a) and b) illustrate how spot requests are fulfilled, where the spot requests are sorted from highest to lowest based on the bid (top to bottom) and are fulfilled based on the number of idle resources. Scenario c) represents arrival of a new spot bid (>spot price), which results in the revocation of lowest bid request. Scenario d) depicts the spot revocation due to loss of the spot supply caused by on-demand and reserved requests.

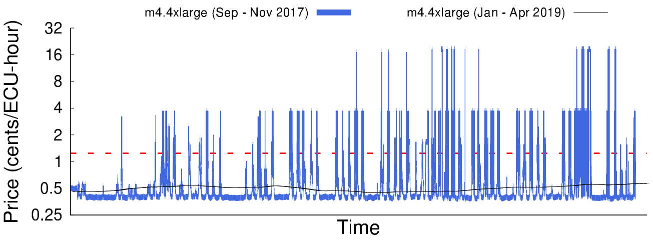


Fig. 3. Comparison of `m4.4xlarge` spot VM price in *us-west-1c* over 3 months both before and after the EC2 spot pricing policy change.

price. The figure shows that before the change spot prices were highly volatile, often spiking to well above the on-demand price, while after the change, spot prices became largely fixed with few variations over the entire 3 month period.

In this paper, we first outline both the advantages and disadvantages of offering transient servers using a real-time spot market (§II). We then discuss similar advantages and disadvantages for the current, largely fixed-price, model that resembles the one from GCE and Azure, and reasons EC2 likely adopted the simpler fixed-price approach (§III). We then discuss possible avenues for addressing the problems with each approach (§IV) before concluding (§V).

II. Real-time Spot Market Pros and Cons

This section discusses the advantages and disadvantages of selling transient servers in a real-time spot market where the spot price is variable, and users bid on idle capacity and can use it as long as their bid exceeds the spot price.

A. *Advantages*

Determines the Optimal Price.. The primary advantage of a real-time spot market is that it automatically determines the price necessary to sell the available supply, which varies over time. If the demand is low or the supply is high, then the price will drop, which, in theory, can attract additional demand that can consume the available supply. In contrast, setting a fixed price on a variable supply may result in periods where demand is lower than supply (resulting in some idle capacity) and where supply is lower than demand (requiring the cloud platform to reject requests for VMs). Both cases reduce the total revenue—-in the former by not selling all of the capacity, and in the latter by selling the idle capacity for too low of a price. The ability for markets to "automatically" set the optimal price is the main reason that market-based allocation of computing resources under constraint has been the subject of research for more than 50 years [3], [4], [36], [5].

Reveals Important Information. EC2 always made the previous 3 months of spot price data available for each of the thousands of spot instances it offered. In addition, there were online archives that stored spot price data for many years. This historical spot price data provided users important information when making resource allocation decisions, as it enabled them to estimate the frequency and distribution of both revocations (at any bid level) and availability. Note that revocations and availability are distinct metrics: revocations represent points in time where the spot price rises above an instance's bid price, causing EC2 to revoke the instance, while availability represents the percentage of time the spot price is below the bid price. Thus, a spot instance can yield the same availability across a wide range of revocation rates and characteristics.

Understanding revocations and availability characteristics is important for applications, and can influence their choice of spot instance and the value they derive from it. For example, a stateless distributed web application that uses a frontend load balancer to distribute requests across a large number of active servers may not care much about the frequency of revocations, since they have little impact on performance (as the load balancer need only update its active server set when a spot instance is revoked) [37], [38]. However, the application would care about availability as it may require a certain target capacity to be available for a certain percentage, e.g., 99.999% available [38]. In contrast, a distributed machine learning job that is iterating on volatile in-memory data may care more about the revocation rate, since it influences the optimal frequency for checkpointing volatile data to disk, i.e., that balances the overhead of checkpointing with the expected time required to re-generate volatile data lost on a revocation [23], [25]. This application may care less about availability, either because it is a background (rather than interactive) workload without a strict deadline or because it always immediately replaces a revoked spot instance with another one.

In both cases above, the applications use knowledge of each spot instance's volatility (or revocation rate) and availability to better optimize their performance. In the case of the stateless distributed web application, it may select different spot instances with independent availability periods to ensure that its aggregate capacity availability requirement is met [38].

In the case of a distributed machine learning job, it may select the spot instance that offers the lowest cost when accounting for the overhead of checkpointing and re-executing work on revocations [39], [23], [25]. Analyzing spot instances prices also enables applications to select different risk tolerances. For example, prior work examines selecting different "portfolios" of spot instances to adjust a distributed application's risk tolerance, since some spot instances offer a low price but high volatility, while others may offer a high price but low volatility [32]. The latter has the potential for lower cost on average across many jobs, but may yield a higher variance in cost for any one job, exposing the application to higher risk.

Prior work on optimizing applications for transient servers generally generally uses information about volatility and availability to select transient servers with different characteristics and to select and tune fault-tolerance mechanisms. For example, Pado focuses on distributed batch jobs that are structured as directed acyclic graphs (DAGs) of tasks, and only executes tasks with few dependencies on transient servers, since a revocation results in a low penalty [22]. SpotOn jointly selects the optimal spot instance and fault-tolerance mechanism, e.g., replication or checkpointing, that offers the lowest cost to complete a batch job based on an application's resource usage. The work shows that both the price characteristics and the resource usage influence the optimal choice [24]. HotSpot chases low prices by continuously migrating applications encapsulated to the globally lowest price spot instance, and shows that doing this also reduces the application's revocation rate [40].

Numerous other works also optimize different applications for spot instances (or transient servers) based on knowledge of price, revocations, and availability. Since spot price data reveals these characteristics, it is critically important for optimizing applications for transient cloud servers.

Control of Revocation and Availability Characteristics. Not only did the spot price reveal revocation rates and availability of spot instances, it enabled applications to control the relative frequency and availability of spot instances by raising or lowering their bid. Thus, applications could "buy" a lower revocation rate and higher availability by raising their bid price. This ability to control revocation rates and availability is important, as different applications may have different tolerances for revocations and availability. This control combined with the ability to select from spot instances with different price characteristics gave users a wide range of options when compiling their "portfolio" of spot instances.

Control of Revocation and Availability Dependencies. The real-time spot price and associated revocation policy also enabled distributed applications useful control over the timing of revocations. Distributed applications often synchronize their state periodically as the application progresses. Many big data frameworks, such as Hadoop and Spark, follow the Bulk Synchronous Processing model [41] such that parallel tasks periodically synchronize at barriers, where all tasks must reach the barrier before the application proceeds. These synchronization barriers degrade performance if progress is non-uniform across tasks as the "fast" tasks end up waiting on the "slow" (or straggler) tasks [39]. These types of distributed batch jobs are attractive for spot instances as they involve bulk data processing that is amenable to delays or performance degradation due to revocations. However, if parallel tasks experience different numbers of revocations, their performance will be bottlenecked by the slowest task (i.e., the one with the most revocations).

EC2's real-time spot market enabled distributed applications to indirectly control the timing of revocations across tasks. Namely, any spot instance with the same bid would experience revocations at the same time based on changes in the common spot price. This control is powerful, as it enables distributed applications to ensure that transient servers experience both the same number of revocations and experience revocations at exactly the same time. Thus, batch applications can maintain uniform performance across parallel tasks even when executing on transient servers. This is not possible under the current spot price model, as revocations are not necessarily correlated with the spot price. Note that while existing models may bulk revoke a single request of, say, N transient servers, there is no way to add servers *post facto* with the same revocation timings. This is simple in the real-time spot market, where you simply make a new request for spot instances with the same bid price. These new spot instances will have revocations at exactly the same frequency and time as other spot instances at this bid price. Thus, EC2's spot market enables a kind of coordinated elasticity for transient servers that enables users to add transient servers with the same revocation pattern.

In addition to controlling the revocation dependencies, the spot market also enables control of availability dependencies. This is critical to running highly available interactive services on spot instances via over-provisioning [38]. In particular, spot instances of the same type requested with the same bid will have exactly correlated periods of availability. In addition, requesting a spot instance of the same type with a higher bid will have availability periods that are a superset of the the availability of the spot instances with a lower bid. To understand why this control is important, consider provisioning a multi-tier web application on spot instances using a container manager, such as Kubernetes [8].

This generally requires specifying the resources for each tier independently. However, if the availability periods are independent for each tier, then the total availability of the service will be the product of the availabilities of each tier. Thus, if each tier is available with percentage p, then the total availability will be p^k for a k-tier service. For example, if 3-tier service has an availability of 99%, then the aggregate availability of the service will only be only $99\%^3 = 97\%$. In contrast, if we can specify each tier such that their availability periods are entirely dependent and correlated, then the 3-tier service availability would simply be the availability of any given tier or 99%. This control enables such interactive services to provide the same availability with many fewer transient servers (at a lower cost).

EC2's real-time spot market enabled sophisticated control of availability and revocation dependency relationships across server requests. As we discuss, the current fixed-price offerings from Google and Microsoft do not.

Always Obtainable. Cloud platforms offer a wide range of different purchasing options, as mentioned in §I, which generally specify a fixed price over some commitment duration. For example, on-demand instances incur a fixed price per unit time and, once allocated, allow users to relinquish them whenever they are done. Similarly, reserved instances incur a fixed price over the length of the reservation, which can vary from 1 to 3 years. EC2 also includes more esoteric purchasing options, such as spot block and scheduled reserve. Spot block enables users to reserve short blocks of time on demand, from 1 to 6 hours, such that the instance is always revoked at the end of the block. Scheduled reserve enables users to reserve repeating blocks of time at daily, weekly, or monthly intervals.

In general, since the price of these offerings does not adjust dynamically to the demand, it is possible for EC2 to run out of these resources. For example, prior work has shown that the real-time price of spot instances (before the change in late 2017) is partially correlated with the availability of on-demand instances [42]. The intuition was that the unavailability of on-demand instances would drive up the price of spot instances, since users could always obtain spot instances by bidding a higher price. That is, the unavailability of on-demand servers would increase the demand (and real-time price) of spot instances. The work actively probed EC2 by requesting on-demand instances during high spot price periods and observed the rate at which the on-demand requests were rejected due to "out of capacity" errors. This rate of on-demand rejection correlated with high spot prices, such that the higher the spot price the higher the on-demand rejection rate.

This insight reveals an important characteristic of spot instances: since their price was dynamic, they were always *obtainable* if users were willing to bid high enough for them. This is not true for fixed price resources. Obtainability is an important, yet often overlooked metric, in public cloud platforms [43]. The metric is important for businesses, which often want assurances that they can obtain cloud resources to satisfy sudden increases in demand. Satisfying sudden increases in demand (likely from new customers) is critical for businesses. With the end of EC2's real-time pricing, the only way to ensure obtainability is on a long-term basis by reserving instances for 1 to 3 year periods. Even then, EC2 may reject requests for reservations to ensure they have enough resources in their on-demand pool to satisfy requests. This can be easily seen in the scheduled reserved option, as some time periods "fill up" for scheduled reserved, such that EC2 prevents additional users from reserving those times. In general, though, users have little visibility into the obtainability of different types of cloud servers.

B. Disadvantages

While EC2's real-time spot market enabled numerous advantages, it also had a number of disadvantages that likely led to its demise. We highlight the important disadvantages below.

Highly Complex. EC2's spot market is highly complex with thousands of server types, each with their own dynamic price. Most users are likely not sophisticated enough to navigate this complexity, and effectively use the information to optimize their applications. While the advantages above are beneficial for sophisticated users in optimizing their applications, most of these advantages have been highlighted through research and are likely not used in practice. Dynamic pricing may also discourage enterprises from using spot instances despite their low average price, as they typically have fixed IT budgets for fixed resources that cannot accommodate variable pricing.

The volatility of the real-time spot market also often caused the spot price to be significantly greater than the on-demand price even if the average price was low. Initially, EC2 had no limit on the bid value, and there were documented instances of the spot price rising to greater than $1000 per hour for instances with an on-demand price of $0.10 per hour (or 10k× higher price). This likely occurred due to convenience bidding [44] where users bid excessively high prices to prevent revocations from occurring under the assumption that the price would never rise significantly above the on-demand price. Of course, if everyone adopts this strategy, it will result in a significant rise in the spot price. After these incidents, EC2 placed bid limits between 4-10× the on-demand price to limit the negative impact of such convenience bidding strategies.

For enterprises that have already had difficulty adapting from budgeting IT as capital costs to recurring cloud costs, variable pricing may have discouraged the use of spot instances. This may be one reason that Google and Microsoft adopted fixed price models for their transient server offerings.

Requires Application Modifications. Using spot instances typically requires application modifications to gracefully handle revocations. While some applications may be designed to handle rare failures, revocations differ from failures in that they are an expected and frequent event, whereas failures are rare and unexpected. Thus, designing applications to handle revocations requires tuning fault-tolerance mechanisms to account for the costs of the mechanism, e.g., the frequency of checkpointing or degree of replication. If users do not account for these costs of revocations or fault-tolerance, then it is possible for spot instances to result in an overall execution cost that is actually higher than on-demand instances, even if they have a lower price. The execution cost accounts for the overhead due to revocation, while the price does not.

Recent work on resource deflation [45] proposes a different model where resources are not simply revoked, but are just reduced to a minimal but still runnable state. This model is easier for applications to handle since it does not subject them to failure-like revocations, but only to performance degradation. However, there are still issues here with handling non-uniform performance, especially for synchronized applications [39]. For example, if resources are not uniformly deflated in a distributed applications that periodically synchronizes its state, it can result in "stragglers" that waste resources by waiting for slow tasks to finish. In the cloud, these wasted resources translate directly to wasted costs.

Ultimately, modifying applications to gracefully handle revocations at low cost is often challenging and application-specific, which may discourage use of spot instances. Interestingly, new serverless offerings which enable users to execute time-limited stateless functions on-demand are similar to transient servers in that they "revoke" the function after some maximum running time, e.g., 300 seconds. However, these offerings, unlike transient servers, *require* developers to re-implement their applications with this usage model in mind.

Not Incentive Compatible. An incentive compatible auction mechanism is one where every user is incentivized to bid according to their true valuation. For example, a sealed-bid Vickrey auction for a single item that charges the (highest) winning bid the bid price of the second highest bid is known to be incentive compatible and elicit truthful valuations from bidders. That is, users gain no advantage from submitting a bid that deviates from their actual truthful valuation. Vickrey-Clarke-Groves (VCG) auctions extend this mechanism to multiple items. EC2 implied (although we do not know) that they used a uniform price auction (possibly with a hidden reserve price [30], [31] at certain times), which compared to VCG auctions incentives bidders of multiple units to bid below their truthful valuation. Of course, EC2's auction was continuous and revealed historical prices, which may also influence bids.

While EC2's auction mechanism may not have been incentive compatible, the more significant problem with eliciting truthful bids is the presence of on-demand instances. Sufficiently adaptive applications have a strong incentive to always bid a value close to the on-demand price. If the spot price exceeds the corresponding on-demand price, then these applications can simply request on-demand instances and switch to using them. Much prior work on optimizing for spot instances adopted this simple bidding strategy [40], [25], [32], [24], [20]. Thus, as applications that use spot instances become more sophisticated, we would expect the real-time spot price to rise to something close to the on-demand price. In fact, EC2 ingrained this bidding strategy into its tools for using spot instances, such as Spot Fleet [46], which by default bids the on-demand price. In addition, after the change in the pricing algorithm that ended the real-time spot price, EC2 replaced the bid with a "maximum price" that is set by default equal to the corresponding on-demand price. Thus, the real-time spot market likely never elicited truthful bids from users.

We highlighted this fact in prior work from 2016 arguing that due to this EC2's real-time spot market was not sustainable [29]. In particular, as applications became more sophisticated, the spot price would not only rise but become more volatile, requiring applications to incur a higher revocation overhead. The higher price and increased overhead would eliminate nearly any benefit to using spot instances. In addition, the presence of a risk-free on-demand price may result in other ways to game the spot market. For example, prior work points out that a low spot price relative to the on-demand price actually implies that a spot instance has a low risk of revocation, since it requires a larger change in the supply/demand balance to trigger the revocation [40]. As a result, the lowest priced spot instances are also the ones with the lowest revocation risk. Thus, dynamically migrating to the spot instance with the lowest price is highly attractive. Of course, this strategy does not work if everyone does this. We discuss these second-order effects on the market below.

Excessive Revocations. The frequency of revocations dictate the inherent value of transient servers and spot instances. The higher the revocation rate, the more overhead applications incur from re-executing lost work or executing fault-tolerance mechanisms. Thus, the revocation rate dictates the usable resources of transient servers. Offering transient servers in a real-time spot markets incurs strictly more revocations than offering them for a fixed price. In the former case, revocations can occur for two reasons: the underlying supply of spot instances changes or the set of bids and values change. In the latter case, revocations can only occur when the supply of spot instances changes, as there are no user bids.

Thus, in a real-time spot market, even if the supply stays the same, the spot price can be highly volatile due to competing users out-bidding each other. As a result, a real-time spot market will experience strictly more revocations than offering transient servers for a fixed price, which decreases the usable resources—not consumed by revocation overhead—and value of the spot instance relative to selling them for a fixed price. Thus, the more volatile the spot market, the less the resources that are sold in the spot market are worth. This dynamic is unlike the market for other commodities, and is another reason why we previously argued that EC2's real-time spot market was not sustainable [29]. Note that these excessive revocations are the "price" that is paid for enabling spot instances to be always obtainable (an advantage from the previous section). To have a resource that is always obtainable, the cloud platform must be able to revoke that resource from other users in response to a user request. Such user-induced revocations cannot occur when using a fixed price.

Second-order Effects.. Many of the proposed cost reducing optimizations for spot instances have potential second-order effects that would likely increase real-time spot prices and make them more volatile if widely adopted, which based on the discuss above would result in more revocations and decrease spot instances' inherent value. For example, the HotSpot system that continuously migrates to the lowest-priced spot instance would increase volatility if everyone adopted it, as all users would chase the same low prices, which would cause that price to rise [40]. The rise would result in another migration that would raise the price of another spot instance. Prior work generally has not considered these second-order effects, since most users of spot instances are currently unsophisticated. The low real-time spot price is direct evidence of unsophisticated users, since if all users were engaged in many of the optimizations mentioned above, the price of spot instances would be closer to on-demand instances.

III. Fixed Price Transient Server Pros and Cons

Since EC2's late-2017 change in its spot pricing algorithm to reflect only "longer-term trends in supply and demand," the spot price volatility has reduced significantly, as reflected in Figure 3. Our analysis across thousands of spot instance prices shows that, while there are some gradual increases and decreases in spot prices over time, the vast majority of spot instances now exhibit very little change in their spot price. In effect, EC2's spot market is now akin to GCE and Azure's fixed-price transient server offerings, as the price is largely stable and does not reflect real-time supply and demand.

In some ways, the current EC2 approach represents the worst of both worlds, as the spot price is technically still dynamic and variable for users, so there is no assurance of a stable price, even if the price is mostly fixed.. In addition, allowing users to issue a default "maximum price" equal to the on-demand price prevents EC2 from seeing users' real value function. While the previous approach did not necessarily elicit truthful bids from all users, it did require users to place a bid (there was no default), and likely provided some limited visibility into user demand. It is likely that the large majority of requests under the current approach just use the default maximum price, regardless of their true valuation. As a result, EC2 likely has less accurate information available to correctly set the long-term spot price. In addition to the disadvantages above, the fixed-price (or near fixed-price model in EC2's case) has advantages and disadvantages that roughly mirror those from the previous section. We elaborate below.

A. Advantages

Less Complex. The fixed-price model is much easier for users to understand and does not expose them to as much complexity compared to real-time spot prices. The model is also easier to budget for enterprises that are used to allocating fixed budgets. This is likely the primary reason that EC2, as well as GCE and Azure, have adopted this model. Less complex offerings are more likely to be used and adopted by customers. Even though the real-time spot price yields the "optimal" price in theory, the price elasticity of demand for cloud servers may not be high, since applications must typically be modified to use transient servers. In general, if applications have been modified and *can* gracefully use transient servers, they *should* use them, since their price is less than the on-demand price. Thus, these applications are not sensitive to the spot price. In contrast, if applications have not been modified to use transient servers, they *cannot* use them regardless of their price.

As a result, deriving the "optimal" price may not be a significant advantage for a real-time spot market, especially since the market does not necessarily elicit truthful bids. Instead, encouraging users to spend the developer time to modify their applications to run on transient servers may be more effective at encouraging their use (and increasing revenue from them). This is likely the primary reason Google and Microsoft have adopted fixed-price transient server offerings.

Incentive Agnostic. The fixed-price model does not require users to make bids, and so is agnostic to incentives and gaming. However, as we discuss below, since users do not know either revocation or availability information, there is less opportunity to optimize applications to efficiently use fixed-price transient servers.

Fewer Revocations. As mentioned in the previous section, in a fixed-price model revocations only occur when the supply of idle capacity changes. There are no revocations related to changes in user demand, i.e., users outbidding other users. As discussed above, fewer revocations is better for applications.

No Second Order Effects. There are no second order effects, since applications have no revocation or availability information to optimize for transient server characteristics. This is an advantage in that users cannot game the market *en masse* in a way that makes transient servers less useful, as can be done with a real-time spot market.

B. Disadvantages

Non-optimal Pricing. The chosen fixed-price is guaranteed to be non-optimal if demand. As a result, the approach may leave excess resources available during periods of low demand, and may require rejecting requests during periods of high demand. However, this disadvantage may be offset due to increased use from less complex pricing. Thus, while in theory, a fixed-price reduces the potential revenue, in practice it may not have a significant impact on the revenue from transient servers.

Requires Application Modifications. The fixed-price model still includes revocations, and thus still requires application modifications. Cloud providers likely view this as the largest impediment to transient server adoption. Simplifying pricing, may encourage developers to address the non-trivial application modifications necessary to use transient servers.

Reveals No Information. The fixed-price model reveals no information about transient server revocation and availability characteristics. As a result, it is impossible to implement most of the optimizations for transient servers proposed in prior work on the fixed-price instances available from EC2, Google, and Azure today. While this prevents gaming and second-order effects, it also results in less efficient applications. For example, a distributed machine learning job that iterates on data stored in volatile memory may checkpoint this state too frequently (or infrequently) resulting in a higher overhead than necessary. Applications also will not be able to effectively choose between different types of transient servers. In the real-time pricing model, applications can see when demand for one type of server rises, and then choose to select another server, which represents a natural form of load balancing.

This lack of information is a major drawback. Prior work has proposed methods for revealing some information about availability and revocation rates [28] that could aid users in optimizing their applications for transient servers. However, unlike with spot instances, this information is not externally verifiable by users. Defining service level objectives (SLOs) that are not externally verifiable by users could potentially lead to users accusing the platform of lying (or to the platform actually lying). As a result, cloud providers often define SLOs that are externally verifiable. In contrast, users can externally verify that their revocation behavior conformed to the real-time spot price by observing when their instances are allocated and revoked relative to their bid and the spot price.

No Control of Revocation and Availability Characteristics.. Not only do fixed-price transient servers not reveal any information about revocation and availability characteristics, they also do not enable any indirect control over them. This eliminates an important dimension of optimization that is available to applications in the real-time spot market.

No Control of Revocation and Availability Dependencies. In EC2, GCE, and Azure there is no way to control the dependency relationships between transient servers allocated in different requests. These transient servers may be revoked at different times and have independent availabilities. As mentioned earlier, many optimizations for distributed applications rely on controlling these dependency relationships.

Not Always Obtainable. As mentioned earlier, fixed-price resources are not always obtainable, as once demand exceeds supply one user cannot take resources from another. Fixed-price transient servers are essentially first-come-first-serve.

IV. Moving Forward

The best way to offer transient cloud servers to users remains an open research question. The spot price model has significant advantages that enable sophisticated applications to optimize their performance. However, the spot market was not sustainable [29] for numerous reasons. In contrast, offering transient servers for a fixed price is simpler (and may encourage more usage), but prevent basic techniques to optimize for transient servers, which decreases application efficiency and transient server value. Thus, a key research question is whether it is possible to combine the advantages of both models. There are two basic approaches for addressing this question: we can either start with the real-time spot price model or the fixed price model and try to fix their respective concerns. We discuss each approach below.

Fixing the Spot Market. The primary drawback of the spot market is that it is highly complex for users. The drawback, which likely motivated EC2 to alter their model, is easily addressed by having software services that allocate resources on applications' behalf. There is ample research into brokers [47], derivative clouds [21], and virtual cloud service providers [48] that mask the complexity of spot instances from applications. Many of these approaches are transparent to applications and implemented at the system [21] or middleware layer [24]. It is also possible to embed market intelligence into applications, so that they can respond to changing market conditions by re-configuring themselves [25], [24], [32]. There are startup companies as well focused on improving the efficiency of cloud usage [49]. These companies are akin to "demand response" companies for the energy sector, except they focus on adjusting the resource usage of their clients' applications in response to changing cloud availability and prices.

Part of the challenge above also intersects the drawback of requiring application modifications. As noted above, it is possible to design approaches that are transparent to applications and implemented at the system layer. In addition, recent work proposes using resource deflation rather than abrupt shutdowns to revoke resources, which has less impact on application correctness [45]. New execution models, such as serverless, also *require* applications to be re-developed as a set of stateless function executions of short duration, e.g., less than 300 seconds. Thus, serverless applications could likely execute on transient servers with few modifications, as the maximum function duration is nearly as small as the revocation warning.

Incentive compatibility is more difficult to fix, as the presence of fixed-price on-demand instances influences bid values. This might require either abandoning the fixed-price model and selling on-demand instances for a variable price, which is highly unlikely for the foreseeable future. However, once software systems are flexible enough to handle revocations, price dynamics, elasticity, and dynamic adaptation, there may be less reason to sell on-demand resources for a fixed price. That said, it depends on the efficiency gains (and peak-to-trough) ratio of cloud usage, which is currently not known. If the peak-to-trough is high, then the gains from dynamic pricing may be significant. Addressing incentive compatibility would go a long way towards addressing both excessive revocations and second-order effects, since these concerns are primarily a function of the lack of incentive compatibility.

Fixing the Fixed-price Model. The fixed-price model cannot address its non-optimal price and lack of obtainability, as these are a function of the fixed price. However, cloud platforms could reveal more information about revocation and availability of different VM types, as proposed in prior work [28]. The challenge here is doing so in a way that is externally verifiable by cloud users without compromising user privacy. The spot pricing model indirectly is able to do this via the spot price. Enabling visibility into revocation and availability characteristics is related to exposing information. In this case, cloud platforms might consider selling different classes of transient servers that have different characteristics for different prices [28]. Finally, enabling control of dependency relationships is more straightforward to address. Cloud platforms could enable users to link requests for transient servers together, such that their revocations or availability periods were concurrent. Of course, these control mechanisms should be designed so that users are not able to game them.

V. Conclusions

This paper discusses the implications of EC2's change in its spot pricing algorithm such that the spot price does not track instantaneous supply and demand, similar to the fixed-price approach of Google and Microsoft. We then compare market-based versus fixed pricing when offering transient cloud servers. The best way to offer transient servers is still an open research question. As a result, we discuss different approaches for fixing the problems with both the spot market approach and the fixed-price approach.

Acknowledgements. This work is funded by NSF grants #1802523, #1815412, #1763834, #1836752, and #1405826, as well as DOD ARL grant W911NF-17-2-019 and the Amazon AWS Cloud Credits for Research program.

REFERENCES

[1] "Spot Instance Product Details," https://aws.amazon.com/ec2/spot/details/, Accessed August 2017.

[2] "Amazon EC2 Beta," https://aws.amazon.com/about-aws/whats-new/2009/12/14/announcing-amazon-ec2-spot-instances/, December 14th 2009.

[3] I. Sutherland, "A Futures Market in Computer Time," *Communications of the ACM*, vol. 11, no. 6, June 1968.

[4] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta, "Spawn: A Distributed Computational Economy," *IEEE Transactions on Software Engineering*, vol. 18, no. 2, February 1992.

[5] J. Shneidman, C. Ng, D. Parkes, A. AuYoung, A. Snoeren, A. Vahdat, and B. Chun, "Why Markets Could (But Don't Currently) Solve Resource Allocation Problems in Systems," in *HotOS*, June 2005.

[6] R. Singh, P. Sharma, D. Irwin, P. Shenoy, and K. Ramakrishnan, "Here Today, Gone Tomorrow: Exploiting Transient Servers in Datacenters," *IEEE Internet Computing*, vol. 18, no. 4, April 2014.

[7] R. Singh, D. Irwin, P. Shenoy, and K. Ramakrishnan, "Yank: Enabling Green Data Centers to Pull the Plug," in *Symposium on Networked Systems Design and Implementation (NSDI)*, April 2013.

[8] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *ACM Queue - Containers*, vol. 14, no. 1, January-February 2016.

[9] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale Cluster Management at Google with Borg," in *European Conference on Computer Systems (EuroSys)*, April 2015.

[10] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-grained Resource Sharing in the Data Center," in *NSDI*, March 2011.

[11] W. Guo, K. Chen, Y. Wu, and W. Zheng, "Bidding for highly available services with low price in spot instance market (hpdc)," in *The International ACM Symposium on High-Performance Parallel and Distributed Computing*, June 2015.

[12] M. Mazzucco and M. Dumas, "Achieving Performance and Availability Guarantees with Spot Instances," in *International Conference on High Performance Computing and Communications (HPCC)*, September 2011.

[13] I. Menache, O. Shamir, and N. Jain, "On-demand, Spot, or Both: Dynamic Resource Allocation for Executing Batch Jobs in the Cloud," in *International Conference on Autonomic Computing (ICAC)*, 2014.

[14] S. Zaman and D. Grosu, "Efficient Bidding for Virtual Machine Instances in Clouds," in *International Conference on Cloud Computing (CLOUD)*, July 2011.

[15] Q. Zhang, E. Gürses, R. Boutaba, and J. Xiao, "Dynamic Resource Allocation for Spot Markets in Clouds," in *Workshop on Hot Topics in Management of Internet, CLoud, and Enterprise Networks and Services (HotICE)*, March 2011.

[16] B. Javadi, R. Thulasiram, and R. Buyya, "Statistical Modeling of Spot Instance Prices in Public Cloud Environments," in *UCC*, December 2011.

[17] Y. Song, M. Zafer, and K. Lee, "Optimal Bidding in Spot Instance Market," in *Infocom*, March 2012.

[18] ——, "Optimal Bidding in Spot Instance Market," in *International Conference on Computer Communications (Infocom)*, March 2012.

[19] L. Zheng, C. Joe-Wong, C. Tan, M. Chiang, and X. Wang, "How to Bid the Cloud," in *ACM SIGCOMM Conference (SIGCOMM)*, August 2015.

[20] P. Sharma, D. Irwin, and P. Shenoy, "How Not to Bid the Cloud," in *Workshop on Hot Topics in Cloud Computing (HotCloud)*, June 2016.

[21] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy, "SpotCheck: Designing a Derivative IaaS Cloud on the Spot Market," in *European Conference on Computer Systems (EuroSys)*, April 2015.

[22] Y. Yang, G. Kim, W. Song, Y. Lee, A. Chung, Z. Qian, B. Cho, and B. Chun, "Pado: A Data Processing Engine for Harnessing Transient Resources in Datacenters," in *European Conference on Computer Systems (EuroSys)*, April 2017.

[23] Y. Yan, Y. Gao, Z. Guo, B. Chen, and T. Moscibroda, "TR-Spark: Transient Computing for Big Data Analytics," in *Symposium on Cloud Computing (SoCC)*, October 2016.

[24] S. Subramanya, T. Guo, P. Sharma, D. Irwin, and P. Shenoy, "SpotOn: A Batch Computing Service for the Spot Market," in *Symposium on Cloud Computing (SoCC)*, August 2015.

[25] P. Sharma, T. Guo, X. He, D. Irwin, and P. Shenoy, "Flint: Batch-Interactive Data-Intensive Processing on Transient Servers," in *European Conference on Computer Systems (EuroSys)*, April 2016.

[26] J. Barr, "Amazon EC2 Update – Streamlined Access to Spot Capacity, Smooth Price Changes, Instance Hibernation," https://aws.amazon.com/blogs/aws/amazon-ec2-update-streamlined-access-to-spot-capacity-smooth-price-changes-instance-hibernation/, November 28th 2017.

[27] D. Chelupati and R. Pary, "New Amazon EC2 Spot pricing model: Simplified purchasing without bidding and fewer interruptions," https://aws.amazon.com/blogs/compute/new-amazon-ec2-spot-pricing/, March 13th 2018.

[28] S. Subramanya, A. Rizk, and D. Irwin, "Cloud Spot Markets are Not Sustainable: The Case for Transient Guarantees," in *HotCloud*, June 2016.

[29] ——, "Cloud Spot Markets are Not Sustainable: The Case for Transient Guarantees," in *HotCloud*, June 2016.

[30] O. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "Deconstructing Amazon EC2 Spot Instance Pricing," *ACM Transactions on Economics and Computation (TEAC)*, vol. 1, no. 3, 2013.

[31] ——, "Deconstructing Amazon EC2 Spot Instance Pricing," in *International Conference on Cloud Computing Technology and Science (CloudCom)*, November 2011.

[32] P. Sharma, D. Irwin, and P. Shenoy, "Portfolio-driven Resource Management for Transient Cloud Servers," in *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, June 2017.

[33] S. Shastri and D. Irwin, "Towards Index-based Global Trading in Cloud Markets," in *Workshop on Hot Topics in Cloud Computing (HotCloud)*, June 2017.

[34] S. Fox, "New AWS Spot Pricing Model: The Good, the Bad, and the Ugly," https://autoscalr.com/2018/01/04/new-aws-spot-pricing-model-good-bad-ugly/, January 4th 2018.

[35] "Google Cloud Platform: Preemptible Virtual Machines," https://cloud.google.com/preemptible-vms/, September 21st 2016.

[36] I. Stoica, H. Abdel-Wahab, and A. Pothen, "A microeconomic scheduler for parallel computers," in *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, April 1995.

[37] A. Ali-Eldin, J. Westin, B. Wang, P. Sharma, and P. Shenoy, "SpotWeb: Running Latency-sensitive Distributed Web Services on Transient Cloud Servers," in *HPDC*, June 2019.

[38] P. Ambati and D. Irwin, "Optimizing the Cost of Executing Mixed Interactive and Batch Workloads on Transient VMs," in *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, June 2019.

[39] P. Ambati, D. irwin, P. Shenoy, L. Gao, A. Ali-Eldin, and J. Albrecht, "Understanding the Synchronization Costs of Distributed ML on Transient Cloud Resources," in *IC2E*, June 2019.

[40] S. Shastri and D. Irwin, "HotSpot: Automated Server Hopping in Cloud Spot Markets," in *SoCC*, September 2017.

[41] L. Valiant, "A Bridging Model for Parallel Computation," *CACM*, vol. 33, no. 8, August 1990.

[42] X. Ouyang, D. Irwin, and P. Shenoy, "SpotLight: An Information Service for the Cloud," in *International Conference on Distributed Computing Systems (ICDCS)*, June 2016.

[43] M. Carvalho, W. Cirne, F. Brasileiro, and J. Wilkes, "Long-term SLOs for Reclaimed Cloud Computing Resources," in *SoCC*, 2014.

[44] J. Boutelle, "What to do when Amazon's spot prices spike, in *Gigaom*," December 27th 2011.

[45] P. Sharma, A. Ali-Eldin, and P. Shenoy, "Resource Deflation: A New Approach for Transient Resource Reclamation," in *European Conference on Computer Systems (EuroSys)*, March 2019.

[46] J. Barr, "New Spot Fleet Option - Distribute Your Fleet Across Multiple Capacity Pools," AWS Blog, https://aws.amazon.com/blogs/aws/new-spot-fleet-option-distribute-your-fleet-across-multiple-capacity-pools/, September 15th 2015.

[47] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. Yocum, "Sharing Networked Resources with Brokered Leases," in *USENIX*, June 2006.

[48] L. Zheng, C. Joe-Wong, C. Brinton, C. Tan, S. Ha, and M. Chiang, "On the Viability of a Cloud Virtual Service Provider," in *SIGMETRICS*, June 2016.

[49] F. Lardinois, "Spotinst, which helps you buy AWS Spot Instances, raises $2M Series A, in *TechCrunch*," March 8th 2016.