The Financialization of Cloud Computing: Opportunities and Challenges

David Irwin, Prateek Sharma, Supreeth Shastri, and Prashant Shenoy
University of Massachusetts Amherst

***Abstract*—Under competitive pressure to maximize their infrastructure's utilization and revenue, modern cloud platforms are quickly evolving into server markets that offer increasingly sophisticated contracts beyond simple on-demand servers, such as spot, preemptible, burstable, and reserved servers. In parallel, continuing advances in system and network virtualization are making server-time a more fungible commodity. These trends have motivated calls for open cloud commodity markets akin to other commodity markets, e.g., for oil, gold, corn, etc. However, such open cloud markets have not yet materialized due to key differences between cloud resources and other commodities. In particular, the relationship between applications and their underlying server resources is fundamentally different and more complex than other commodities. Unfortunately, software developers generally do not have the necessary background to effectively manage this complexity as part of their applications. Financial cloud computing is an emerging area that focuses on adapting and extending concepts from economics and finance to explicitly manage applications' tradeoffs between cost, risk, availability, and performance in cloud markets. A key goal of financial cloud computing is to develop systems-level abstractions and mechanisms that manage the market's complexity. This paper introduces this emerging area and its potential benefits, surveys related work, discusses challenges to realizing a cloud commodity market, and then outlines future research directions.**

I. Introduction

Cloud computing is quickly becoming the foundation of our Information-based economy, providing the large-scale computing power necessary to advance nearly every segment of society, including transportation, energy, communication, healthcare, science, and entertainment. Cloud platforms provide numerous benefits, including low prices, a pay-as-you-go billing model, the on-demand allocation of resources, and the illusion of near-infinite scalability. As a result, businesses are increasingly renting computation and storage from the cloud rather than using their own private infrastructure.

Cloud platforms have evolved significantly over the past ten years. Initially, platforms exposed only a simple abstraction to users: the choice of a small number of *on-demand* servers for a fixed per-hour price. However, under competitive pressure to maximize their infrastructure's utilization and revenue, modern cloud platforms are quickly evolving into complex markets for server-time by offering a multitude of increasingly sophisticated *contracts* beyond simple on-demand servers that specify much more complex Service Level Objectives (SLOs) [1]. For example, spot instances in EC2 enable users to bid on servers. If a user's bid price exceeds the servers' current spot price, the platform allocates the servers to the user, who pays the variable spot price for them. However, if the spot price rises above the bid price, the platform immediately *revokes* the servers after a brief warning [2]. While spot servers expose users to a high revocation risk, they cost ∼50-90% less than on-demand servers, which platforms cannot revoke. The spot market exposes key elements that are intrinsic to all markets—risk and uncertainty—that are largely been absent from (or hidden by) platforms that offer only on-demand servers. While the risk of spot servers is clear, as we discuss, other contracts expose users to other risks that are less apparent.

By offering different contracts, cloud platforms can increase their infrastructure's utilization and revenue, while also offering users more customized purchasing options tailored to their requirements. For example, spot instances enable platforms to allocate their idle capacity, while retaining the flexibility to reclaim those resources to satisfy higher-priority requests, e.g., for on-demand or reserved servers. In parallel, spot instances benefit users with flexible, delay-tolerant workloads that can tolerate revocations by significantly reducing their costs. In contrast, reserved instances in EC2 require users to purchase server-time upfront for a long fixed term, e.g., 1 to 3 years, and includes an "obtainability" guarantee [3]. That is, EC2 guarantees to never reject a request for a previously reserved instance due to lack of capacity. Reserved instances enable highly risk-averse users to ensure capacity is available when necessary. Note that, since on-demand instances do not have an obtainability guarantee, EC2 may reject requests during periods of high demand. Thus, on-demand services implicitly expose users to *rejection risk*. In addition to the contracts above, EC2 and other platforms now offer a wide range of other contract variants that differ in their cost model, price level, performance guarantees, and risk exposure [1], [4].

The sophistication and diversity of cloud contracts increasingly mirrors those found in other *commodity markets*, e.g., for fuel, electricity, agricultural products, and metals, which offer a range of similar contract types. Companies, such as electric utilities and airlines, that rely on these commodities typically engage in sophisticated market strategies to explicitly manage their tradeoff between the commodity's cost and its risk, e.g., of future price volatility and supply shortages. Similarly, cloud platforms represent an emerging commodity market for server-time that technology companies will increasingly have to actively engage in to compete, as companies that can significantly reduce their costs, while limiting their risk, will gain a competitive advantage. In essence, computing

ICCCN17; Vancouver, CA; August 2017
XXX-X-XXXX-XXXX-8/17/$31.00 ©2017 IEEE

represents the new "fuel" for the IT economy.

The trends above have motivated calls for an open cloud commodity market from both researchers [5], [6], [7] and industry [8], [9], [9], [10], [11], [12], [13], [14], [15], [16], [17]. However, such open cloud commodity markets have not yet materialized due to key differences between cloud resources and other commodities. In particular, the relationship between applications and their underlying server resources is fundamentally different and more complex than other commodities. For example, servers are stateful such that migrating between servers, or experiencing a server revocation in the spot market, incurs a performance penalty and may affect application correctness. In addition, applications have numerous, often implicit, dependencies on a specific platform based on its underlying hardware, software, and network, which restricts the flexibility of applications to "move with the market" and always exploit the lowest cost resources. Finally, there are security and privacy concerns with running applications in the cloud, requiring the user and provider to trust each other.

As a result, server-time has not yet to become a truly *fungible* resource, such that a server from one provider can freely substitute for a server from another provider. The complexity above also creates significant barriers to entering the market, resulting in commodity clouds currently being dominated by a small number of large, monopolistic providers, e.g., Amazon, Google, and Microsoft. Thus far, only these few providers have been able to amass the technical expertise, trust, and scale required to offer cloud servers as a commodity.

However, continuing advances in system and network virtualization are increasingly making server-time more fungible. For example, recent work on nested virtualization and "superclouds" is eliminating many of the implicit dependencies that bind applications to a specific cloud or hardware platform [18]; advances in data center and wide-area networking are reducing the time and performance penalty from migrating state to take advantage of low prices; and new security and privacy mechanisms, such as Intel SGX, are decreasing the trust applications must place in cloud providers and vice versa [19], [20]. In addition, the emergence of large-scale co-location data centers, which host cloud servers on behalf of customers, are enabling small providers to take advantage of the operational efficiencies and economies-of-scale of large data centers [7].

As servers become more fungible, we expect cloud server resources to evolve into a true commodity market where a wide range of providers can offer server resources under a variety of pricing models. That said, the scale of today's monopolistic cloud platforms has already spawned rich internal cloud markets for resources, as indicated by the large set of complex contracts that are now offered. Unfortunately, application developers generally do not have the background to manage the complexity of optimally selecting from a large number of complex cloud contracts as part of their applications. As a result, most applications still use simple on-demand servers, despite their relative high cost, with a recent report indicating that cloud users are not highly sensitive to server prices [9]. In particular, cloud servers have a low price elasticity of demand, such that changes in price have a small effect on the demand.

This price inelasticity has motivated a "race to the top," where cloud providers add increasingly higher level services to attract consumers, rather than reduce server prices. The common assumption is that users do not respond to server prices because they do not care. However, we argue that users are not sensitive to changes in price because applications are not "financially aware:" their policies do not treat cloud resource allocation as an investment decision, and their mechanisms and policies are not sufficiently flexible to enable them to respond to changes in market prices. As spot instances illustrate, for applications that can effectively manage this complexity, cloud markets offer the potential for significant savings.

Financial cloud computing is a new area that focuses on adapting and extending concepts from economics and finance to automate and simplify the management of applications' cost and risk in cloud markets. Importantly, financial cloud computing differs from prior work on "market-based" resource allocation, which i) largely targeted synthetic markets using virtual currency and not real money [21], ii) focused on optimizing resource allocation and not managing risk, and iii) pre-dated the emergence of complex server contracts offered by real cloud platforms. In particular, a key goal of financial cloud computing is to elevate risk management to a first-class application and systems design principle by selecting the set of servers and contracts that yield the highest risk-adjusted returns. As we discuss, we can apply risk management techniques across multiple dimensions, e.g., time, server types, contracts, etc., and at multiple layers of the software stack, e.g., the application-, systems-, and financial-level.

This paper surveys recent work on developing financially-aware cloud applications and systems, and discusses future research directions. Section 2 reviews background and related work that falls under the umbrella of financial cloud computing. Section 3 then describes core risk management techniques from finance, including diversification, trading, and hedging, and initial adaptations and extensions of these techniques for cloud computing platforms. Section 4 discusses applying these techniques at different levels of the software stack. Finally, Section 5 outlines potential research directions and concludes.

II. Background

Similar to conventional financial services, financial cloud computing determines how a user (or group of users) should invest in cloud resources, e.g., by selecting the mix of servers and contracts required to meet a target level of performance for an expected workload. In conventional commodity markets, there are a range of spot contracts and futures contracts. Spot contracts offer commodities for immediate delivery, while futures contracts offer commodities for delivery at some future date. Companies that rely on commodities buy and sell spot and futures contracts based on their expectations of future workload and future market prices. For example, an airline might purchase fuel futures to meet its expected fuel demands over the next year. Then, as the year progresses, if the actual demand is lower or higher than the expected demand at any

Contract	*Provider*	*Cost Model*	*Price Level*	*Performance*	*Rejection Risk*	*Revocation Risk*
On-demand	EC2	Fixed Hourly	Fixed (Medium)	Low Variability	Yes	No
On-demand	EC2	Fixed Hourly	Fixed (Medium)	High Variability	Yes	No
Dedicated	EC2	Fixed Hourly	Fixed (High)	Fixed	Yes	No
Dedicated	EC2	Fixed Yearly	Fixed (High)	Fixed	Yes	No
Reserved	EC2	Fixed Yearly	Variable (High)	Low Variability	No	No
Reserved Market	EC2	Variable	Variable (High)	Low Variability	No	No
Spot	EC2	Variable Hourly	Variable (Low)	Low Variability	Subject to Bid	Variable
Spot Block	EC2	Variable 1-6 hours	Variable (Low)	Low Variability	Subject to Bid	On Block Expiry
On-demand	GCE	Sustained-use Discount	Fixed (Medium)	Low Variability	Yes	No
On-demand	GCE	Sustained-use Discount	Fixed (Medium)	High Variability	Yes	No
Preemptible	GCE	Fixed	Fixed (Low)	Low Variability	Yes	Yes

TABLE I: *Summary of the high-level terms for a subset of the server contracts offered by EC2 and GCE. Contract terms differ in their cost model, price level, performance, rejection risk, and revocation risk [1], [4].*

point, it might sell or buy, respectively, spot contracts (or shorter term futures contracts) to ensure they match their real-time supply with their demand. In general, the more accurate a company's predictions of its future demand and future market prices, the higher the savings it can achieve in the market.

However, there are many differences between cloud commodity markets and conventional commodity markets. In particular, while the investment strategies in conventional commodity markets consist of pure financial transactions, any trading of cloud resources must directly integrate with the systems and applications running on those resources. For example, to use a recently-allocated spot server, users must first initialize an application on it, which introduces system and application complexity and incurs a performance penalty. Cloud resources are also not a homogenous pool like other commodities, but are instead divided into discrete resource bundles, e.g., of CPU, memory, disk, etc., based on the underlying server hardware with a distinct price. As we discuss, cloud applications have an incentive to actively "trade" these resource bundles, e.g., by migrating from one bundle to another bundle, as their resource demands change to minimize their cost per resources used. However, cloud resource demands change in real time, and are much more dynamic than most other commodities. Server-time is similar to electricity in that it must be used in real time, or it is wasted. To take advantage of such trading, though, systems must be flexible enough to migrate to new resources.

As the examples above illustrate, financial cloud computing must not only address the strategies for investing in cloud resources (based on expectations of future workload and prices), similar to other commodities, but also the systems mechanisms required to exploit the underlying resources.

A. Contract Overview

Table I highlights the current complexity of cloud contracts by listing 11 different server contracts currently offered by two popular cloud platforms (EC2 and GCE), and how they differ in their cost model, price level, performance guarantees, and risk exposure [1], [4]. Note that the table offers only a coarse approximation of each contract's terms, as they are too complex to precisely capture in a single table. In general, these contracts offer different tradeoffs between cost and risk, such that the lower the risk exposure the higher the cost. Unfortunately, categorizing cloud contracts is challenging, since the contracts mix and match numerous arcane options. We outline the basic options below and their tradeoff. Note that, for simplicity, we do not include all available options.

We list seven distinct contract types in Table I.[1] On-demand servers are the simplest contract, incurring a fixed price per unit time, e.g., every hour or minute, and enabling users to request and relinquish them at anytime. Reserved servers differ from on-demand servers in that they incur a much higher upfront cost for a 1- or 3-year term, and have no *rejection risk*. By contrast, platforms may periodically reject new requests for on-demand servers [22]. However, reserved servers eliminate much of the elasticity benefits of using the cloud for variable workloads, and may significantly increase costs if not highly utilized. To provide some elasticity for reserved servers, platforms also operate reserved server marketplaces, where users can recoup their upfront costs—if they are not utilizing reserved servers as expected—by selling the remaining term of their reservations. The price of reserved servers in this market varies based on supply and demand, and is similar to a futures market, where users lock in a price for future resources.

Spot servers are perhaps the most direct example of dynamism in the cloud market. Spot servers generally offer low prices ($\sim$50-90% less than on-demand) but expose users to a high *revocation risk*, since a price spike that causes a revocation can occur at anytime. Interestingly, spot servers have a higher obtainability than on-demand servers, since their price is market-driven: if users bid high enough, they are nearly guaranteed to acquire a spot server. In prior work, we show that users can acquire spot servers even during periods when requests for on-demand servers are rejected [22].

The global spot market is massive, as EC2 operates a separate spot market with its own dynamic spot price for each server type and configuration in each Availability Zone (AZ) of each region.[2] Importantly, there is often a difference in the price of the same server in different AZs, and the normalized price per unit of resource of different servers in the same AZ. Such price differentials in the market represent a potential arbitrage opportunity that financially-aware applications can exploit to lower their costs. For example, Figure 1(top) is an

[1] We repeat some contract types to reflect their different options.

[2] Different AZs are akin to separate data centers.

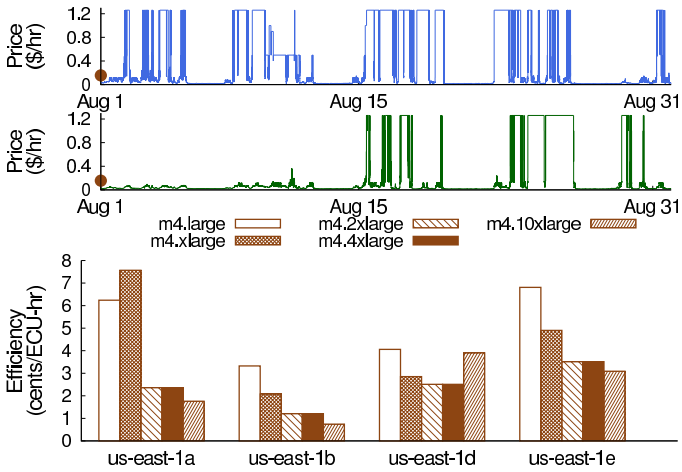


Fig. 1: *The spot price of the* `m4.large` *server varies across two AZs of EC2's US-East region over 8/16 (top). The normalized efficiency in $/ECU of the* `m4` *family of servers varies across spot markets within each availability zone of the US-East-1 region over 8/16 (bottom). The red dot indicates the on-demand price. Analysis and figure from [23].*

analysis from recent work [23] that shows the `m4.large` server's price in two AZs of EC2's US-East-1 region over August 2016 differs by up to 30×. Similarly, Figure 1(bottom) (also from [23]) shows the average normalized price per EC2 Compute Unit (ECU), which defines a relative measure of a server's integer processing power [24], for the five different servers in the `m4` server family of the same AZ differs by up to 4× over the same period. These price differentials represent a potential arbitrage opportunity that financially-aware applications can exploit to lower their costs. Similarly, preemptible VMs in GCE also have a low price and high revocation risk, but instead incur a fixed price.

Spot and preemptible servers represent a type of *transient server* that is available for an uncertain amount of time, and may be revoked at anytime. Since their introduction [25], [26], there has been significant research on optimizing system performance on transient servers in many contexts [25], [26], [27], [28], [29], [30], [31]. As we discuss in prior work [31], transient servers require the use of fault-tolerance mechanisms to gracefully handle revocations. Since these fault-tolerance mechanisms incur an overhead, the performance and value of transient servers is inherently less than that of stable on-demand servers. Spot-block contracts were recently introduced by EC2 to mitigate the impact of revocations on performance by enabling users to bid for fixed blocks of time between one and six hours. If allocated, the platform guarantees to revoke spot-block servers only at the end of their time block but not before. Since spot-block servers have no revocation risk over their time block, they are worth more than spot servers, e.g., costing ∼50%, and do not require the continuous use of high-overhead fault-tolerance mechanisms.

Finally, users may select from servers offered under the contracts that are either burstable or not burstable, such that they have either high or low performance variability, respectively [32]. The latter allocates cores to VMs and exhibits low performance variability (only due to cross-talk among co-located VMs), while the former allows VMs to fairly share cores, such that performance varies based on the utilization of co-located VMs. To eliminate performance variability, platforms also offer dedicated servers on isolated hardware. In general, since performance variability is a form of risk, the more variable the performance the lower the cost, e.g., dedicated servers cost more than servers with low performance variability, which cost more than those with high performance variability. Variable performance has a similar affect on cost-efficiency as variable spot prices, as in both cases, the cost per unit of resource consumed is variable.

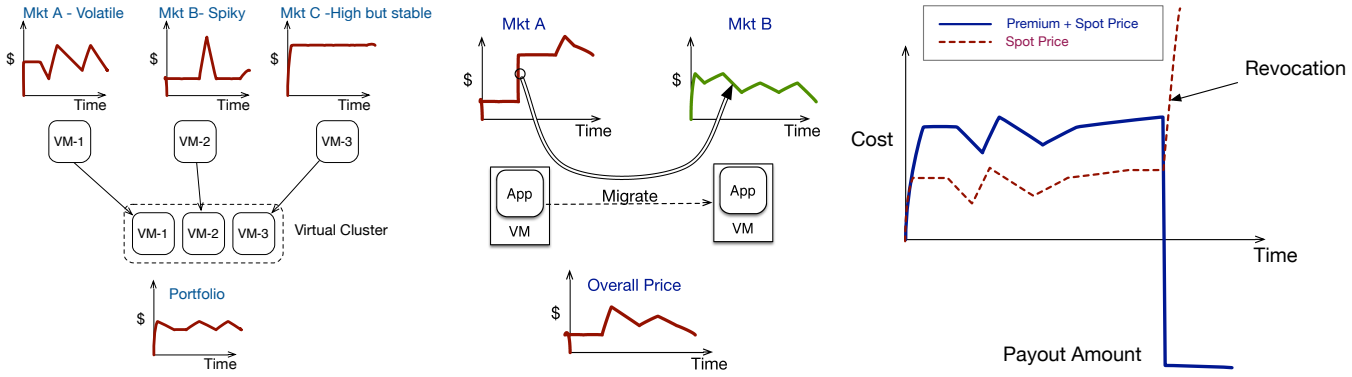
B. Related Work

There is a long history of prior research on market-based resource allocation prior to the emergence of cloud computing, e.g., [33], [34], [35], [36], [37], [21], [38], [39]. Limited space precludes surveying the entire body of work. However, prior work is not applicable to today's cloud markets, as it focused on i) optimizing resource allocation in synthetic markets using virtual currency, ii) did not address risk management in modern distributed applications, and iii) did not envision the diversity and complexity of cloud server contracts.

More recently, there has been a variety of work on exploiting arbitrage in the spot market by both researchers [40], [41], [29], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52] and industry [53], [54], [55]. However, this prior work only focuses on spot servers and does not leverage the wide range of other contracts offered by cloud platforms. Thus, this prior work does not consider the full range of "investment" strategies available to users. In addition, by focusing on spot servers, the allocation policies are inherently short-term and do not consider longer term investment strategies, e.g., using reserved servers. Prior work also often focuses on optimizing the use of spot servers for only narrow classes of applications [40], [29], [41] that necessitates complex application-specific modifications [42], [29], [28], [47], [48], [30].

Financial cloud computing is broader than the work above in that it focuses on developing general platforms and tools that simplify development across a wide range of financially-aware applications, which leverage the full range of contracts offered by cloud platforms (potentially over much longer time horizons). While some recent work examines other cloud contracts, it focuses on "gaming" their inefficiencies and not simplifying the development of market-aware applications. For example, Zheng et al. [56] exploit GCE's billing model, which offers a sustained-use discount that reduces a server's per-hour price the longer a user holds it, by enabling users to hold servers indefinitely and resell their resources to other consumers. Likewise, HCloud attempts to select contracts based on predictions of servers' performance variability [32].

In addition, there have also been recent proposals (largely in industry) for forming an exchange for trading financial derivatives on cloud resources, e.g., futures, swaps, etc. [8], [9], as well as some initial research on such financial instru-



(a) Diversification (b) Active Trading (c) Hedging

Fig. 2: *Depiction of cloud risk management strategies—diversification, active trading, and hedging—adapted from finance.*

ments [5], [57], [58], [59]. However, unlike financial cloud computing, these proposals treat server resources exactly like other commodities, such as electricity, fuel, gold, corn, etc., and do not consider i) the impact of risk on a running applications' performance and availability or ii) the complexity of developing automated applications that are market-aware.

III. Risk Management Techniques

Financial cloud computing adapts and extends concepts from finance to manage various forms of risk in cloud markets. In particular, Figure 2 depicts three general techniques for managing risk in finance: *diversification*, *active trading*, and *hedging*. Below, we discuss these techniques and how they can be adapted and extended to cloud platforms.

A. Diversification

Diversification reduces risk by investing in a variety of assets, such that the aggregate portfolio of the combined assets has less volatility than any one of the individual assets. Figure 2(a) depicts how the aggregate price of a diversified portfolio can be less volatile than its underlying assets. In constructing a diversified portfolio, the goal is to maximize its *risk-adjusted returns*, i.e., the returns per the level of risk exposure. Note that the absolute returns on a diversified portfolio may be less than the returns on a portfolio that includes only a single asset, but with less risk (or variability) in the expected outcome. For example, in the cloud spot market, a distributed application that prioritizes minimizing cost may opt to select all servers from the cheapest spot market. While this approach may incur the lowest cost, it exposes the application to a high risk of revocation, since a single price spike in the market may trigger the concurrent revocation of all servers. If revocations actually occur, then an application may run for a longer period that causes it to incur a much higher cost in some cases. In contrast, diversifying server selection across multiple spot servers with higher, but uncorrelated, prices, may increase the expected cost but lowers the risk of concurrent revocations (and thus the probability of incurring a high cost).

Applications can apply diversification across multiple dimensions. The example above diversifies *across servers* in different spot markets. Similarly, applications can diversify *across contracts* by adapting the mix of reserved, on-demand, and spot servers they use over time. As we discuss below, adapting the mix of contracts is related to active trading of contracts and hedging long-term versus short-term risk. While there is a large body of work on portfolio construction in finance, this work generally derives from Modern Portfolio Theory (MPT) [60], which is a mathematical framework for maximizing expected returns for a given level of risk exposure. In recent work, we adapted MPT to guide spot server selection in cloud markets [61]. We summarize this MPT cloud adaptation below as one example of adapting and extending existing diversification techniques from finance to the cloud.

MPT defines the risk-adjusted returns for a portfolio of investments as the expected return across all the investments, e.g., based on history, discounted by the portfolio's weighted risk, where the weight α defines the investor's risk tolerance: $E[Return] - \alpha \cdot Risk$. The expected return is the average return of each of n investments in the portfolio weighted by the amount of each investment held. MPT captures each investment's returns using a vector $\mathbf{c} = [\text{Return}_1, \ldots, \text{Return}_n]$. Similarly, we let x_i denote the fraction of each investment i held in a portfolio ($0 \leq x_i \leq 1$). Given this, MPT computes the expected return of a portfolio as $E[Return] = \mathbf{c}\mathbf{x}^T$, where $\mathbf{x} = [x_1, \cdots x_n]$ denotes the portfolio allocation vector.

MPT captures a portfolio's risk based on the likelihood of correlated prices among its investments. MPT defines risk *across many investments* by quantifying the likelihood of correlated prices among the investments. In a low-risk portfolio, if one investment's price spikes, the other investments' prices are unlikely to experience a concurrent spike, dampening the impact of individual price fluctuations. MPT computes risk by defining a covariance matrix $\mathbf{V}$ that captures pairwise correlations between all investment pairs, where V_{ij} is the correlation between investments i, j. Thus, in MPT, $Risk = \mathbf{x}\mathbf{V}\mathbf{x}^T$.

For a given set of investments and α, this MPT formulation defines the optimization problem below that is solvable using any off-the-shelf convex solver, such as CPLEX [62]. The solution to MPT defines an *efficient frontier* of optimal portfolios that offer the highest expected return for a given level of risk (or, equivalently, the lowest risk for a given level of expected return). While there are many possible portfolios on the efficient frontier, which differ in the level of risk they

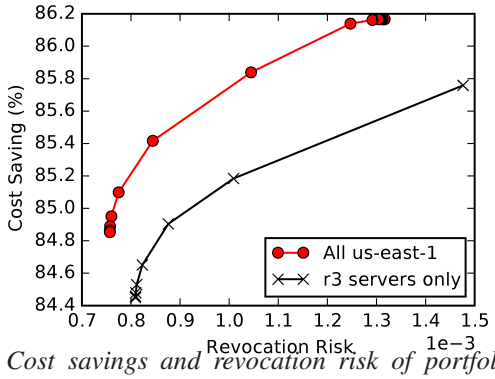


Fig. 3: *Cost savings and revocation risk of portfolios with different risk tolerances. Selecting a portfolio from a larger collection of servers (all US-east-1 vs. only* r3 *servers) results in higher returns at lower risk. Figure from [61].*

expose to users, portfolios that lie below the efficient frontier do not provide enough return for their level of risk.

$$\text{Maximize:} \quad \mathbf{c}\mathbf{x}^T - \alpha \mathbf{x}\mathbf{V}\mathbf{x}^T \tag{1}$$

$$\text{Subject to:} \quad \sum_{1}^{n} x_i = 1$$

$$\mathbf{x} \geq 0$$

To adapt MPT to the cloud, we consider each spot server in our application's "portfolio" to represent an "investment." Likewise, each spot servers' "return" can be computed relative to the "risk free" return defined by the equivalent on-demand server's price. Thus, for a spot server in market i, $Return_i = 1 - \frac{E[S_i]}{D_i}$ where D_i denotes the on-demand cost and $E[S_i]$ denotes the average spot price of market i. Of course, spot servers are not investment holdings in a traditional sense, as applications only rent time on servers and do not sell them at a later date. In particular, the primary "risk" in using spot servers is that of mass revocations due to correlated price increases, which cause a high performance penalty. Thus, rather than compute the covariance matrix of pure spot prices, we instead compute the covariance matrix of revocations [61].

Figure 3 demonstrates the benefits of intelligent portfolio selection by computing the efficient frontier for different risk tolerances of portfolios drawn from different sets of cloud servers. As the figure shows, the expected cost savings increase as applications take on additional risk. In one case, we construct a portfolio from every cloud server available in EC2's `us-east-1` region, and in the other case we restrict the portfolio selection to only the `r3` server types in the `us-east-1` region. As the figure shows, the former results in a 1% increase in cost savings, but a 20-50% reduction in revocation risk, i.e., the probability of all servers in the portfolio being concurrently revoked. This reduction occurs because choosing from a larger set of candidate markets increases the number of markets with low correlations.

Note that our current MPT extension does not consider the effect of revocations on application performance. As discussed earlier, the performance impact of revocations can be mitigated by employing fault-tolerance techniques, which in-turn incur their own overhead that can increase costs or decrease performance. Thus, it is important to select and tune fault-tolerance techniques based on the expected server volatility to optimize cost and performance. For example, prior work shows that different fault-tolerance techniques, such as replication and checkpointing, have different overheads, depending on an application's resource usage and price characteristics [29]. In addition, recent work also looks at optimizing checkpointing for different applications to balance its performance overhead and benefits [42], [30], [27], [28]. In some cases, this work takes advantage of characteristics of specific applications, such as the DAG structure of many "big data" frameworks [27].

Our current MPT adaptation i) does not consider the effect of fault-tolerance mechanisms on the optimal portfolio and ii) focuses purely on spot and on-demand servers. We are considering these extensions as part of future work. An efficient frontier that considers fault-tolerance mechanisms is likely different and application-specific. We can also apply similar principles from finance to diversify our holdings across other types of contracts in Table I. For example, we can balance the use of reserved, spot, and on-demand servers based on long-term expectations of our workload. Purchasing reserved contracts can "lock in" prices in the future if the future workload is well-known. Even if the future workload changes, we can sell the remaining term on reservations in a futures market. EC2 operates such a reserved marketplace where the remaining term on reservations are offered for a variable price. We could also leverage variable performance servers by translating variations in performance to variations in price per unit of resource purchased. In addition, supporting multiple applications introduces additional possible extensions, as there is an opportunity to carve up larger servers among applications, e.g., using resource containers.

B. Active Trading

Since both cloud market prices and application resource usage are highly dynamic and change in real time, the cost per unit of resource consumed by an application on a server is highly dynamic. Thus, at start time, while an application might select the cheapest available server to execute, over the course of its execution, market conditions might change such that another server is cheaper (or less risky). Financial cloud computing should enable applications to exploit these cheaper prices by "trading" resources and proactively migrating to the cheaper host. Virtualized platforms capable of such transparent "trading," including nested virtual machines [63], [64] and resource containers [65], [66], are now widely available. Figure 2(b) shows how trading at the "right" time can reduce costs. In this case, the application trades a VM in market A for a VM in market B just as market A's price spikes, enabling it to maintain a low overall price throughout its execution.

The diversification techniques above only select from servers offered by the primary cloud market to include in its portfolio based on their historical price and volatility characteristics. Thus, the techniques do not consider actively trading servers as market conditions change. Such high-frequency active trading servers provides applications another dimension

by which to reduce their risk. For example, by always trading for the server with the lowest price, relative to its on-demand price, applications can nearly eliminate the risk of revocation, since lower prices (relative to the on-demand price) indicates low risk. Of course, trading to minimize risk presents a tradeoff with other potential objectives. For example, the lowest risk server may not be the most cost-efficient (in terms of price per resources consumed) or the most performance-efficient (in terms of eliminating performance bottlenecks). Thus, applications must define *trading policies* that define their tradeoff between risk, cost, and performance.

There are many possible trading policies for cloud servers. In finance, there are standard measures for assessing the risk-adjusted returns of a single asset, similar to the MPT approach above. For example, the Sharpe Ratio is the most common measure: for an asset i, it is the ratio of the expected difference between the asset's returns R_i and the risk-free returns R_{free} divided by the standard deviation of the returns σ_i. As above, the risk free price is captured by a server's on-demand price. Essentially, the Sharpe ratio quantifies returns relative to risk exposure. Note that many similar metrics have been proposed in finance for assessing the risk-adjusted price of a single asset, such as the Treynor, Sortino, Sterling ratios, etc.

$$S_i = \frac{E[R_i - R_{free}]}{\sigma_i} \tag{2}$$

Thus, trading policies could rank markets based on these ratios and then trade servers to ensure an application runs on the highest ranked server. Of course, these ratios only capture price risk, and do not consider other important metrics for cloud servers, such as resource usage and performance. For example, an application may execute a trade if its workload demands increase, and the resources of its current server are not adequate. In addition, "trading" servers incurs some overhead to migrate state from one server to another. In some cases, this overhead may cause downtime and unavailability, e.g., if live migration is not available. Thus, applications may need to rate-limit their trades to ensure a target level of availability. In general, we must adapt the approaches above for financial cloud computing, to consider not only the price level and its volatility, but also performance and availability.

Active trading policies must also consider the different overheads required to transition within data centers (or AZs), between data centers, and across wide geographical regions. For example, trading a server in the U.S. East region for a server in Asia, which requires migrating state, will incur a much larger overhead than trading for another server within the U.S. East region. In recent work, we show that the selection of a data center (or AZ) and region based on the average price across all servers in the AZ or region can be just as important as selecting the individual server market that maximizes the risk-adjusted returns [67]. This is akin to investors that not only make investment decisions based on individual stock prices, but also on the characteristics of broad market indices, such as the S&P 500, NASDAQ, and Dow Jones. Thus, an application may not choose the globally optimal server if it resides in an AZ or region that has a high overall price or volatility. One advantage of making decisions based, in part, on these broader market indices is that they tend to be more stable and predictable than individual spot markets. While there has been significant prior work on modeling and predicting individual spot market prices in EC2, there has been little work on characterizing price indices for different availability zones, regions, and server classes. However, our experience suggests that, for flexible applications capable of trading, the characteristics of these broader price indices are more important than the characteristics of individual servers.

Spot Market	Revocations (per day)
c3.2xlarge.vpc.1a	22.6
g2.2xlarge.vpc.1e	21.8
g2.8xlarge.1a	18.0
m4.large.vpc.1d	17.6
m3.xlarge.vpc.1d	17.6
m4.xlarge.vpc.1a	16.4
r3.4xlarge.1a	15.8
g2.2xlarge.vpc.1a	15.7
c3.2xlarge.vpc.1e	15.7
c3.xlarge.1a	15.2
Trading Policy	**7.5**

TABLE II: *A risk-averse trading policy can achieve a lower revocation rate than any primary server by migrating to the server with the lowest risk of revocation. Table from [23].*

By actively trading servers based on changes in price, we can effectively define new "derivative" cloud servers that have different cost, risk, performance, and availability characteristics than any server in the primary cloud market. For example, Table II from recent work [23] simulates the benefits of a trading policy that continuously migrates to the lowest risk server. In this case, we only consider migrating among a subset of the ten most volatile markets in EC2 with the highest revocation rates. The figure shows the average revocation rate for these volatile spot servers over August 2016, as well as the average revocation rate achieved by the simulated trading policy as their spot prices fluctuate. The table demonstrates that a trading policy can create a derivative cloud server with a revocation rate over $2\times$ less than any single server in the primary cloud market, even the one with the lowest absolute revocation rate. Thus, in this case, by transparently trading resources we created a new "derivative" server with much lower risk than any server in the primary market. Table III shows a similar result for a trading policy that continuously migrates to the lowest cost server among the servers listed. As above, the "derivative" server created by the cost-based trading policy has a $>2\times$ lower cost than any primary server.

In addition, such active trading approaches could be combined with the MPT-based diversification strategies above. In effect, these derivative servers introduce a nearly infinite number of server options for MPT that were not previously available to select from. However, computing the new efficient frontier based on the new set of derivative servers defined by various trading policies introduces a significant challenge, especially when considering that the "optimal" risk

Spot Market	Efficiency (¢/ECU-hr)
m1.small.1d	44.00
m1.small.vpc.1d	43.99
m1.large.vpc.1d	43.75
m1.large.1d	43.75
m1.xlarge.vpc.1d	43.75
m1.xlarge.1d	43.74
g2.8xlarge.vpc.1d	24.99
m3.medium.1c	23.33
m3.medium.vpc.1c	23.33
d2.2xlarge.1b	22.42
Automaton	**10.89**

TABLE III: *A cost-efficient trading policy can achieve a lower cost than any primary server by migrating to the server with the current lowest cost. Table from [23].*

tolerance on an application's efficient frontier also depends on the overhead of the fault-tolerance mechanisms applications employ to mitigate the effect of revocations. Thus, determining how to select and tune trading policies and fault-tolerance mechanisms to construct a diversified application-specific portfolio represents a significant research challenge. In addition, the ultimate benefits of such approaches relative to applying diversification in the primary market is an open question.

C. Hedging

The diversification and trading techniques above integrate directly with applications and systems. These techniques require adapting applications to optimize the tradeoff between cost, risk, performance, and availability. However, some applications may not be easily adaptable. For example, to recover from spot server revocations, applications often periodically checkpoint their state, and then resume from the last checkpoint. However, not all applications are amenable to such periodic checkpointing. Since, for these applications, restarting may be the only viable option, spot server revocations may result in financial losses, since applications must still pay for revoked servers even though their cycles were wasted.

We can manage risk for these applications by adapting and extending another technique from finance: hedging. A hedge is an investment that offsets the potential losses (or gains) from another investment. For example, airlines and electric utilities often hedge against volatility in future fuel and electricity prices by purchasing futures contracts, which guarantee payment and delivery of fuel or electricity for a set price at a future date. Hedging strategies differ widely based on the type of investment, and are typically done through pure financial instruments, such as futures contracts, short selling, etc. While the diversification strategies above may affect application performance and availability, hedging in cloud markets enables applications to offset any potential financial losses from using the market, and thus provides an orthogonal application-independent way to manage cloud risk.

To understand hedging, consider the analogy of a user buying a home or a car: they also purchase insurance to protect their financial investment. If the car or home is damaged, e.g., due to an accident or a fire, then the insurance can pay for the loss. A similar analogy holds for businesses purchasing a hedge (in the form of a financial option) on a commodity, such as fuel or an agricultural product. If the price of this commodity rises above a threshold set by the option, then the option covers all the "losses" incurred by the price increase. Many businesses, such as airlines, frequently hedge on commodities, to guard against price spikes.

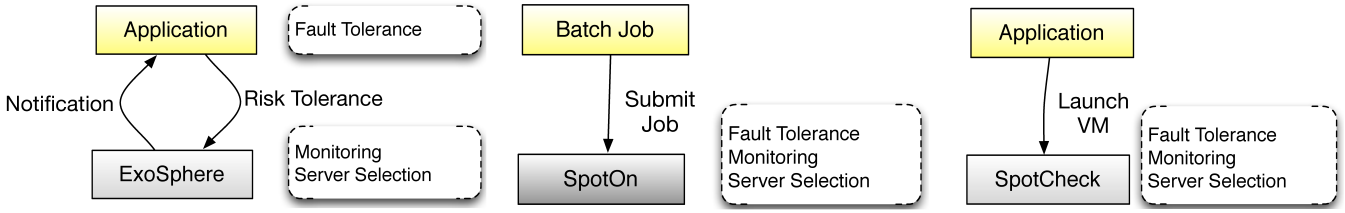
Insurance and option markets spread the risk of a single 'bad event" over time. With insurance, users pay regular premiums which increase the effective cost of their investment, but gain a buffer against catastrophic events by getting a refund on their investment. For the insurance provider, a large pool of users allows it to spread *its* risk by ensuring that it has to refund only a small fraction of customers at any given time.

We can apply similar types of hedging techniques to cloud servers using the same principle. Consider a user that leases N spot servers or preemptible VMs. Suppose that the servers get revoked after time t. Then the user must still pay $N \times t \times Price$ to the cloud operator. If the jobs running on these servers are not amenable to checkpointing, this represents a complete loss, since the jobs must be restarted from the beginning, and the computation done prior to the revocation is wasted.

To avoid such losses, a third-party cloud provider could offer "hedged cloud servers," which are built using revocable spot or preemptible servers, but are priced differently. Specifically, the cost of a hedged server is higher than a regular revocable server, where the price difference is equivalent to the server's "insurance premium." If the platform revokes the server, then the user is given a full or partial refund of their bill, similar to how an insurance policy pays out on the loss of an insured product. Figure 2(c) depicts the cost of a hedged server, where, prior to a revocation, the user pays an additional premium for the server. However, the insurance policy pays out when the revocation occurs, allowing the user to recoup any resulting financial losses. Such hedged cloud servers could specify many different options that offer different levels of protection over different time horizons, similar to different values and terms on life insurance policies.

Of course, "pricing" such options requires analyzing the revocation probability of spot servers and the pool of customers requesting protection. As with insurance, the goal is to keep the premiums low so that it is attractive for customers to "insure" their cloud servers from revocation, while also ensuring that the refunds do not exceed the total premiums. As an example, below we illustrate a simple initial option pricing scheme for single-server cloud jobs running on spot servers.

For each job, we take the historical Mean-Time-Between-Failure (MTBF) of each spot server (based on historical spot price traces) and the job's estimated running time L, to compute the expected probability p that the job's spot server is revoked over its duration, where $p = \frac{L}{MTBR}$. Given this revocation risk, the option pays out a refund R if a revocation actually occurs. Since the refund payouts must be less than the premium revenues over the long run, the following inequality must hold: $\Pi \geq \frac{p}{R} = \frac{L}{MTBR} \cdot R$. Thus, the premium depends on the refund amount R, the $MTBR$, and the job's running time, and may be adjusted by the user. For example, a user



(a) Application-aware Design (b) Application-specific Design (c) Application-independent Design

Fig. 4: *Three different points in the design space for financially-aware systems: an application-aware design that requires application modifications, an application-specific design that supports a particular class of applications (in this case, batch jobs), and an application-independent design that is entirely transparent and supports any application.*

might set a refund amount equal to the total cost of running the job on the server. We have analyzed the average price and MTBF across 1500 EC2 spot markets. Our analysis indicates that premiums are low in most cases, since the MTBR of most spot markets is ∼100 hours. Thus, for a ten hour job, the premium's cost is only 5% of the average spot server price. Since spot servers are typically 50-90% cheaper than on-demand servers, this premium overhead for spot servers is still small relative to their potential savings.

D. Exploiting Multiple Providers

While we discuss the risk management techniques within the context of a single provide, in this case EC2, above, applying the techniques across multiple providers has many benefits. Since different providers offer different contracts at different prices, the use of multiple providers opens up additional optimization opportunities. For example, rather than offer a reserved instance, Google offers a sustained usage discount on on-demand instances that have been operating for an extended period of time. Likewise, as mentioned earlier, Google offers preemptible instances—its version of spot instances—for a fixed price. Thus, applications can consider the presence of these contracts when composing a diversified portfolio or considering trading opportunities. As an application's resource usage varies, the "optimal" server contract or set of servers may change from one provider to another. Of course, there are potential overheads to using multiple providers that must be considered for such optimizations, such as the time to migrate state between providers (or replicate input data), and configuring networks to maintain connectivity.

IV. Financial Cloud Computing Platforms

The risk management techniques above focus on different policies for selecting server (and contracts) that balance risk, cost, performance, and availability tradeoffs. Financial cloud computing platforms implement these policies in conjunction with other important functions, including resource and price monitoring and fault-tolerance. Each of these functions can be implemented at either the application-level or systems-level of the software stack. For example, much of the prior work on optimizing for spot instances in EC2 integrates market-aware optimizations directly into the application. As a result, these optimizations are tailored to each specific application and are not applicable to other applications. In contrast, the risk management techniques in the previous section are general and applicable across a wide range of applications. General risk management strategies permit systems-level implementations that can either apply to broad classes of applications or be entirely application-independent. Below, we summarize three different design points—application-aware, application-specific, and application-independent—depicted in Figure 4 from our recent work, as well as discuss other possible designs.

A. Application-aware Design

In recent work, we designed a platform for portfolio-driven resource management on transient servers, called ExoSphere [61]. Similar to other cluster managers, like Mesos and Kubernetes, ExoSphere supports a large class of data-parallel applications, such as Spark and Hadoop, on cloud platforms. Importantly, ExoSphere decouples the server selection policy from the application by automatically selecting from the available set of spot servers using the MPT formulation from the previous section. As a result, ExoSphere relieves individual applications (and developers) from monitoring market prices and optimizing server selection. However, as discussed in the previous section, there are certain aspects of running on spot servers that are inherently application-specific, such as the choice of fault-tolerance mechanisms to handle revocations.

To support such application-specific choices, ExoSphere enables application's to register handlers that it asychronously upcalls at regular intervals or when specific market events occur, such as a spike in prices or a server revocation. These upcalls enable applications to adjust their operation or portfolio risk tolerance based on changes in market conditions. For example, applications can select and configure their own application-specific fault-tolerance mechanism, e.g., checkpointing or replication, to mitigate the loss of state on a revocation. We have used ExoSphere to port multiple applications, including Spark, Hadoop, and MPI jobs, to efficiently use spot servers on EC2. Exosphere's application-aware design is depicted in Figure 4(a), as it requires application modifications to respond to market events.

B. Application-specific Design

Figure 4(b) depicts the architecture of SpotOn [29], a batch scheduler that acquires its resources from EC2's spot market. This represents another point in the design space where the system transparently supports a particular class of applications,

in this case batch jobs, without requiring any modifications. This architecture enables the system to select the spot servers and the underlying fault-tolerance mechanism, e.g., replication or checkpointing, in concert to optimize application performance and cost. Since the design is transparent to applications, it does not permit applications to react to unexpected changes to market conditions or workload demands, which may decrease the savings relative to ExoSphere's application-aware approach. However, the advantage of this design is that application's require no modifications. The difference is akin to similar problems in operating system design, which must either reveal or hide resource allocation decisions from upper-level applications: while approaches that reveal resource allocation decisions can improve performance, they can also increase application complexity. ExoSphere adopts an "exokernel" approach that reveals changes in market conditions that applications can respond to, while SpotOn adopts a more traditional approach that hides resource allocation decisions.

C. Application-independent Design

Finally, Figure 4(c) shows SpotCheck [68], which represents an application-independent design that supports any application by attempting to mask the complexity of the underlying market. Similar to derivative servers, SpotCheck defines a derivative cloud, which repackages and resells resources purchased from native IaaS platforms under contract terms tailored to a specific class of user. As with derivative servers, a derivative cloud can offer resources with different pricing models and availability guarantees not provided by native cloud platforms using a mix of resources purchased under different contracts. Derivative clouds completely decouple "investment" decisions in cloud resources from the higher-level application decision to create virtualized instances, e.g., using resource containers or nested VMs. That is, applications can create as many virtualized instances as they wish, while the derivative cloud decides how to acquire cloud resources and map the virtualized instances onto them.

SpotCheck is a derivative cloud that provides availability guarantees similar to on-demand instances, but for a price near that of spot instances. Essentially, SpotCheck migrates away from spot servers upon revocation (with the help of a backup server) to maintain availability. While SpotCheck enables completely transparent use of spot instances (unlike the two previous designs), it comes at a cost in both performance and price. In particular, SpotCheck must maintain an additional backup server to store live memory state to handle revocations at an additional cost, such that each application must commit all writes to the backup server in real-time to ensure it can recover on a revocation. This overhead adds an additional performance cost relative to the other designs above.

D. Other Design Points

There are other designs possible, including hybrids. For example, a hybrid design could actively manage the ratio of reserved, spot, and on-demand servers based on expectations of future workload demands and prices without attempting to entirely mask revocations from users like SpotCheck, but instead provide upcall interfaces similar to ExoSphere on a revocation. Another design could support multiple broad classes of applications, such as batch and interactive applications. In this case, interactive applications might require allocating many more spot servers than necessary to achieve a target level of available capacity with high probability to ensure low latency responses, thereby leaving some resources idle much of the time. Since spot servers have low prices, this approach will still be cheaper than allocating on-demand servers. In this case, the background tasks could execute "for free" on the variable amount of idle capacity. Cluster managers, such as Mesos and Kubernetes, already support a similar binary distinction between background and foreground tasks.

V. Conclusion

This paper introduces the emerging area of financial cloud computing and outlines its potential benefits. Financial cloud computing focuses on adapting and extending concepts from economics and finance to explicitly manage applications' tradeoffs between cost, risk, availability, and performance in cloud markets. Importantly, rather than define application-specific market optimization techniques as in prior work, financial cloud computing focuses on supporting more general policies and systems that support broad classes of applications. We outline three general risk management strategies from finance and discuss how we have adapted and extended them to support cloud resources in recent work. For each general risk management strategy, we also discuss additional extensions and optimization opportunities, as well as the potential to combine them. We then discuss different designs for financial cloud computing that integrate risk management at different layers of the software stack, and detail the tradeoffs for each.

Acknowledgements. This work is supported by the NSF (#1422245) and a Google Faculty Research award.

References

[1] "Amazon EC2 Pricing," https://aws.amazon.com/ec2/pricing/, November 14th 2016.

[2] J. Barr, "New - EC2 Spot Instance Termination Notices," https://aws.amazon.com/blogs/aws/new-ec2-spot-instance-termination-notices/, January 6th 2015.

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

[4] "Google Compute Engine Pricing," https://cloud.google.com/compute/pricing, November 14th 2016.

[5] A. Toosi, R. Thulasirămă, and R. Buyya, "Financial Option Market Model for Federated Cloud Environments," in *International Conference on Utility and Cloud Computing (UCC)*, November 2012.

[6] A. Bestavros and O. Krieger, "Towards an Open Cloud Marketplace: Vision and First Steps," *IEEE Internet Computing: View from the Cloud*, January 2014.

[7] P. Desnoyers, J. Hennessey, B. Holden, O. Krieger, L. Rudolph, and A. Young, "Using Open Stack for an Open Cloud Exchange (OCX)," April 2015.

[8] R. Cohen, "Compute Derivatives: The Next Big Thing In Commodities?" Forbes, October 2nd 2013.

[9] "451 Research, Cloud Price Index Report," https://451research.com/cloud-price-index-overview, November 2015.

[10] B. Butler, "Researchers: Cloud is no commodity, It's a race to the top, not the bottom," http://www.networkworld.com/article/3077247/cloud-computing/researchers-cloud-is-a-no-commodity.html, May 31st 2016.

[11] T. Worstall, "Cloud Services Become, Quite Literally, A Commodity," https://www.forbes.com/sites/timworstall/2014/04/15/cloud-services-become-quite-literally-a-commodity/#3e94a5ec1a9a, April 15th 2014.

[12] M. Cooter, "Trading cloud services," http://www.datacenterdynamics.com/content-tracks/colo-cloud/trading-cloud-services/93595.fullarticle, March 20th 2015.

[13] ——, "Data Center Dynamics, trading Cloud Services," http://www.datacenterdynamics.com/content-tracks/colo-cloud/trading-cloud-services/93595.fullarticle, March 20th 2015.

[14] R. Bissett, "Wired, utilities, commodities, and the evolution of iaas markets," http://insights.wired.com/profiles/blogs/utilities-commodities-and-the-evolution-of-iaas-markets#axzz4cjwj4GLx, October 8th 2014.

[15] U. Dauer, "German Stock Exchange to Offer Trading in Cloud Computing," The Wall Street Journal, July 2nd 2013.

[16] "The Economist, A Market for Computing Power," http://www.economist.com/node/18185752, February 17th 2011.

[17] J. Novet, "VentureBeat, How are cloud contracts like soybean futures? Both are tradeable commodities," April 14th 2014.

[18] Z. Shen, Q. Jia, E. Sela, B. Rainero, W. Song, R. van Renesse, and H. Weatherspoon, "Follow the Sun through the Clouds: Application Migration for Geographically Shifting Workloads," in *SoCC*, October 2016.

[19] A. Baumann, M. Peinado, and G. Hunt, "Shielding Applications from an Untrusted Cloud with Haven," in *OSDI*, December 2014.

[20] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Securing Linux Containers with Intel SGX," in *OSDI*, December 2016.

[21] 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*, May 2005.

[22] X. Ouyang, D. Irwin, and P. Shenoy, "SpotLight: An Information Service for the Cloud," in *ICDCS*, June 2016.

[23] S. Shastri and D. Irwin, "Automaton: Enabling Automated Resource Trading in Cloud Markets," University of Massachusetts Amherst, Tech. Rep., March 2017.

[24] C. Babcock, "Amazon's 'Virtual CPU'? You Figure it Out," December 23rd 2015.

[25] R. Singh, D. Irwin, P. Shenoy, and K. Ramakrishnan, "Yank: Enabling Green Data Centers to Pull the Plug," in *NSDI*, April 2013.

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

[27] Y. Yang, G. Kim, W. Song, Y. Lee, and A. Chung, "Pado: A Data Processing Engine for Harnessing Transient Resources in Datacenters," in *EuroSys*, April 2017.

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

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

[30] A. Harlap, G. R. Ganger, and P. B. Gibbons, "TierML: Using Tiers of Reliability for Agile Elasticity in Machine Learning," Carnegie Mellon University, Tech. Rep., May 2016.

[31] S. Shastri, A. Rizk, and D. Irwin, "Transient Guarantees: Maximizing the Value of Idle Cloud Capacity," in *SC*, November 2016.

[32] C. Delimitrou and C. Kozyrakis, "HCloud: Resource-Efficient Provisioning in Shared Cloud Systems," in *ASPLOS*, April 2016.

[33] C. Waldspurger, T. Hogg, and B. Huberman, "Spawn: A Distributed Computational Economy," *IEEE Transactions on Software Engineering*, vol. 18, no. 2, pp. 103–117, February 1992.

[34] I. Sutherland, "A Futures Market in Computer Time," *CACM*, vol. 11, no. 6, June 1968.

[35] I. Stoica, H. Abdel-Wahab, and A. Pothen, "A Microeconomic Scheduler for Parallel Computers," in *Workshop on Job Scheduling Strategies for Parallel Processing (JSPP)*, April 1995.

[36] D. Irwin, J. Chase, L. Grit, and A. Yumerefendi, "Self-recharging virtual currency," in *EconP2P*, August 2005.

[37] D. Irwin, L. Grit, and J. Chase, "Balancing Risk and Reward in a Market-based Task Service," in *HPDC*, June 2004.

[38] F. Popovici and J. Wilkes, "Profitable Services in an Uncertain World," in *SC*, November 2005.

[39] A. AuYoung, L. Grit, J. Wiener, and J. Wilkes, "Service Contracts and Aggregate Utility Functions," in *HPDC*, June 2006.

[40] B. Huang, N. Jarrett, S. Babu, S. Mukherjee, and J. Yang, "Cumulon: Matrix-Based Data Analytics in the Cloud with Spot Instances," *PVLDB*, vol. 9, no. 3, November 2015.

[41] Z. Xu, C. Stewart, N. Deng, and X. Wang, "Blending On-Demand and Spot Instances to Lower Costs for In-Memory Storage," in *Infocom*, July 2016.

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

[43] Q. Jia, Z. Shen, W. Song, R. van Renesse, and H. Weatherspoon, "Smart Spot Instances for the Supercloud," in *CrossCloud*, April 2016.

[44] W. Voorsluys and R. Buyya, "Reliable Provisioning of Spot Instances for Compute-Intensive Applications," in *AINA*, 2012.

[45] I. Menache, O. Shamir, and N. Jain, "On-demand, Spot, or Both: Dynamic Resource Allocation for Executing Batch Jobs in the Cloud," in *ICAC*, 2014.

[46] S. Khatua and N. Mukherjee, "Application-centric Resource Provisioning for Amazon EC2 Spot Instances," in *EuroPar*, August 2013.

[47] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See Spot Run: Using Spot Instances for MapReduce Workflows," in *HotCloud*, 2010.

[48] H. Liu, "Cutting MapReduce Cost with Spot Market," in *HotCloud*, June 2011.

[49] S. Yi, D. Kondo, and A. Andrzejak, "Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud," in *CLOUD*, July 2010.

[50] M. Mattess, C. Vecchiola, and R. Buyya, "Managing Peak Loads by Leasing Cloud Infrastructure Services from a Spot Market," in *HPCC*, September 2010.

[51] Q. Zhang, E. Gürses, R. Boutaba, and J. Xiao, "Dynamic Resource Allocation for Spot Markets in Clouds," in *HotICE*, March 2011.

[52] O. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "Deconstructing Amazon EC2 Spot Instance Pricing," in *CloudCom*, 2011.

[53] "Batchly, Inc." http://www.batchly.net/, September 2016.

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

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

[56] 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.

[57] B. Sharma, R. K. Thulasiram, P. Thulasiraman, and R. Buyya, "Clabacus: A Risk-adjusted Cloud Resources Pricing Model using Financial Option Theory," *IEEE Transactions on Cloud Computing*, vol. 3, no. 3, pp. 332–344, 2015.

[58] A. N. Toosi, K. Vanmechelen, K. Ramamohanarao, and R. Buyya, "Revenue Maximization with Optimal Capacity Control in Infrastructure as a Service Cloud Markets," *IEEE Transactions on Cloud Computing*, vol. 3, no. 3, pp. 261–274, 2015.

[59] S. Qanbari, F. Li, S. Dustdar, and T.-S. Dai, "Cloud Asset Pricing Tree (CAPT)-Elastic Economic Model for Cloud Service Providers." in *CLOSER*, 2014, pp. 221–229.

[60] H. Markowitz, "Portfolio Selection," *The journal of finance*, vol. 7, no. 1, pp. 77–91, 1952.

[61] P. Sharma, D. Irwin, and P. Shenoy, "Portfolio-driven Resource Management for Transient Cloud Servers," in *SIGMETRICS*, June 2017.

[62] "Risk-return trade-off," http://cvxopt.org/examples/book/portfolio.html, 2016.

[63] M. Ben-Yehuda, M. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B. Yassour, "The Turtles Project: Design and Implementation of Nested Virtualization," in *OSDI*, October 2010.

[64] D. Williams, H. Jamjoom, and H. Weatherspoon, "The Xen-Blanket: Virtualize Once, Run Everywhere," in *EuroSys*, April 2012.

[65] "Linux Containers," http://linuxcontainers.org.

[66] G. Banga, P. Druschel, and J. Mogul, "Resource Containers: A New Facility for Resource Management in Server Systems," in *OSDI*, February 1999.

[67] S. Shastri and D. Irwin, "Towards Index-Based Global Trading in Cloud Spot Markets," in *HotCloud*, June 2017.

[68] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy, "SpotCheck: Designing a Derivative IaaS Cloud on the Spot Market," in *EuroSys*, April 2015.