Lifting the Haze off the Cloud: A Consumer-Centric Market for Database Computation in the Cloud

Yue Wang Alexandra Meliou Gerome Miklau

College of Information and Computer Sciences
University of Massachusetts Amherst
{yuewang,ameli,miklau}@cs.umass.edu

ABSTRACT

The availability of public computing resources in the cloud has revolutionized data analysis, but requesting cloud resources often involves complex decisions for consumers. Estimating the completion time and cost of a computation and requesting the appropriate cloud resources are challenging tasks even for an expert user. We propose a new market-based framework for pricing computational tasks in the cloud. Our framework introduces an *agent* between consumers and cloud providers. The agent takes data and computational tasks from users, estimates time and cost for evaluating the tasks, and returns to consumers *contracts* that specify the price and completion time. Our framework can be applied directly to existing cloud markets without altering the way cloud providers offer and price services. In addition, it simplifies cloud use for consumers by allowing them to compare contracts, rather than choose resources directly. We present design, analytical, and algorithmic contributions focusing on pricing computation contracts, analyzing their properties, and optimizing them in complex workflows. We conduct an experimental evaluation of our market framework over a real-world cloud service and demonstrate empirically that our market ensures three key properties: (a) that consumers benefit from using the market due to competitiveness among agents, (b) that agents have an incentive to price contracts fairly, and (c) that inaccuracies in estimates do not pose a significant risk to agents' profits. Finally, we present a fine-grained pricing mechanism for complex workflows and show that it can increase agent profits by more than an order of magnitude in some cases.

1. INTRODUCTION

The availability of public computing resources in the cloud has revolutionized data analysis. Users no longer need to purchase and maintain dedicated hardware to perform large-scale computing tasks. Instead, they can execute their tasks in the cloud with the appealing opportunity to pay for just what they need. They can choose virtual machines with a wide variety of computational capabilities, they can easily form large clusters of virtual machines to parallelize their tasks, and they can use software that is already installed and configured.

Proceedings of the VLDB Endowment, Vol. 10, No. 4

Yet, taking advantage of this newly-available computing infrastructure often requires significant expertise. The common pricing mechanism of the public cloud requires that users think about low-level resources (e.g. memory, number of cores, CPU speed, IO rates) and how those resources will translate into efficiency of the user's task. Ultimately, users with a well-defined computational task in mind care most about two key factors: the task's completion time and its financial cost. Unfortunately, many users lack the sophistication to navigate the complex options available in the cloud and to choose a configuration[1] that meets their preferences.

As a simple example, imagine users who need to execute a workload of relational queries using the Amazon Relational Database Service (RDS). They need to select a machine type from a list of more than 20 possible options, including "db.m3.xlarge" (4 virtual CPUs, 15GB of memory, costing \$0.370 per hour) and "db.r3.xlarge" (4 virtual CPUs, 30.5GB of memory, costing \$0.475 per hour). The query workload may run more quickly using db.r3.xlarge, because it has more memory, however the hourly rate of db.r3.xlarge is also more expensive, which may result in higher overall cost. Which machine type should the users choose if they are interested in the cheapest execution? Which machine type should they choose if they are interested in the cheapest execution completing within 10 minutes? Typical users do not have enough information to make this choice, as they are often not familiar with configuration parameters or cost models.

The reality of users' choices is even more complex since they may choose one of five data management systems through RDS, or other query engines using EC2, including parallel processing engines, and different configuration options for each. They might also be tempted to compare multiple service providers, in which case they would have to deal with different pricing mechanisms in addition to different configuration options. Amazon RDS charges based on the capacity and number of computational nodes per hour; Google BigQuery charges based on the size of data processed; Microsoft Azure SQL Database charges based on the capacities of service tiers like database size limit and transaction rate.

As a result of this complexity, many users of public clouds make naïve, suboptimal choices that result in overpayment, and/or performance that is contrary to their preferences (e.g., it exceeds their desired deadline or budget). Thus, instead of paying only for what they need, the reality is that they pay for what they do not need and, even worse, they pay more than they have to for it.

[1]A configuration here means a set of system resources and its settings, provided by the cloud provider. It includes the number of virtual instances of a cluster, the buffer size of a cloud database, and so on.

A market for database computations

To ease the burden on users we propose a new market-based framework for pricing computational tasks in the cloud. Our framework introduces an entity called an *agent*, who acts as a broker between consumers and cloud service providers. The agent accepts data and computational tasks from users, estimates the time and cost for evaluating the tasks, and returns to consumers *contracts* that specify the price and completion time for each task.

Our market can operate in conjunction with existing cloud markets, as it does not alter the way cloud providers offer and price services. It simplifies cloud use for consumers by allowing them to compare contracts, rather than choose resources directly. The market also allows users to extract more value from public cloud resources, achieving cheaper and faster query processing than naive configurations. At the same time, a portion of the value an agent helps extract from the cloud will be earned by the agent as profit.

Agents are conceptually distinct from cloud providers in the sense that they use intelligent models to estimate time and cost given consumers queries. In other words, agents take the risk of estimation, while service providers simply charge based on resource consumption, which guarantees profit. In practice, an agent could be a service provider (who provides estimation as a service in addition to cloud resources), a piece of software sold to consumers, or a separate third party who provides service across multiple providers.

Scope. Our goal in this paper is *not* to develop a new technical approach for estimating completion time or deriving an optimal configuration for a cloud-based computation. Prior work has considered these challenges, but, in our view, has not provided a suitable solution to the complexity of cloud provisioning. The reason is that estimation, even for relatively well-defined tasks like relational workloads, is difficult. Proposed methods require complicated profiling tasks to generate models and specialize to one type of workload (e.g., Relational database [20] or MapReduce [1, 16]). In addition, there is inherent uncertainty in prediction, caused by multi-tenancy common in the cloud [35, 21]. Lastly, users' preferences are complex, involving both completion time [33] and cost [50, 24], which have been considered as separate goals [16, 27], but have not been successfully integrated.

Our market-based framework incentivizes expert agents to employ combinations of existing estimation techniques to provide this functionality as a service to non-expert consumers. Users can express preferences in terms of their *utility*, which includes both time and cost considerations. Uncertainty in prediction becomes a risk managed by agents, and included in the price of contracts, rather than a problem for users. Ultimately, our work complements research into better cost estimation in the cloud [46, 8, 16]. In fact, our market will function more effectively as such research advances and agents can exploit new techniques for better estimation.

In this paper, we make the following contributions:

- We define a novel market for database computations, including flexible contracts reflecting user preferences.
- We formalize the agent's task of pricing contracts and propose an efficient algorithm for optimizing contracts.
- We perform extensive evaluation on Amazon's public cloud, using benchmark queries and real-world scientific workflows. We show that our market is practical and effective, and satisfies key properties ensuring that both consumers and agents benefit from the market. Further, we demonstrate that our agent-based market performs better and has fewer restrictions compared to benchmark-based and game-theoretic alternatives.

Outline. We present the market overview and main actors in Section 2, define contracts and optimal pricing in Sections 3 and 4, support fine-grained pricing to further optimize contracts in Section 5. In Section 6, we present a thorough evaluation of our market, and demonstrate that it guarantees several important properties and outperforms alternatives. Finally, we discuss related work and extension and summarize in Sections 7, 8 and 9, respectively.

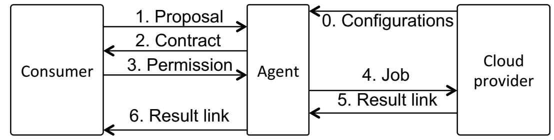


Figure 1: An overview of the participants' interactions in the computation market: the consumer, the agent and the cloud provider.

2. COMPUTATION MARKET OVERVIEW

In this section, we discuss the high-level architectural components of our computation market: 3 types of participants and their interactions through computation contracts. Our computation market exhibits several desirable properties as we mention in Section 2.3.

2.1 Market participants

Our goal is to model the interactions that occur in a computation market, and design the roles and framework in a way that ensures that the market functions effectively. Our computation market involves three types of participants:

Cloud provider. Cloud providers are public entities that offer computational resources as a service, on a pay-as-you-go basis. These resources are often presented as virtual machine types and providers charge fees based on the capabilities of the virtual machines and the duration of their use. Our framework does not enforce any assumptions on the types, quantity, or quality of resources that a cloud provider offers.

Consumer. A consumer is a participant in our computational market who needs to complete a computational task over a dataset D. We assume the computational task is a set of queries or MapReduce jobs[2], denoted as $\mathbf{Q} = \{Q_1, ..., Q_n\}$. We assume that the consumer does not own the computational resources needed to complete $\mathbf{Q}$, and thus needs to use cloud resources. However, the consumer may not have the expertise to determine which cloud provider to use, which resources to lease, or how to configure them. In our framework, the consumer wishes to retrieve the task results $\mathbf{Q}(D) = \{Q_1(D), ..., Q_n(D)\}$ within a specified timeframe, and pay for these results directly. Therefore, the consumer's goal is to complete the task efficiently and for low cost. Different consumers have different time and cost preferences. They will describe these preferences precisely using a utility function, as described later in Section 3.1.

Agent. Consumers' needs are task-centric (time and price to complete a given task), whereas cloud providers' abilities are resource-centric (time and price for a type of resource). Due to this disparity, consumers and providers do not interact directly in our framework. Rather, a semantically separate entity, the *agent*, is tasked with handling the interactions between consumers and cloud providers. The agent receives a task request from a consumer and, in response, calculates a price to complete the task, providing the consumer with a formal contract. We review contracts in Section 2.2, and describe them in detail in Section 3. The agent executes accepted contracts using public cloud resources, and earns a profit whenever the contract price is greater than the actual cost of executing the task. The agent's goals are to attract business by pricing contracts competitively and to earn a profit with each transaction. One of the main

[2]For simplicity of terminology we use "query" to refer to either a query or MapReduce job.

challenges for the agent is to assign accurate prices to consumer requests, which requires knowledge of cloud resources, their capabilities and costs, and expertise in tuning and query prediction.

Figure 1 illustrates the interactions among the 3 market participants. In step 0, the agent collects details on available configurations from the cloud provider to derive later price quotes on consumers' requests. This step may only need to be initiated once, and reused afterwards. In steps 1 through 3, the agent receives a proposal including $\mathbf{Q}$ and statistics about dataset D, denoted s_D, which are sufficient for pricing. For example, s_D can be the number of input records in each table [4], histograms on key columns or sets of columns [45], a small sample of data [16], and other standard statistics. The agent reasons about possible configurations and estimates the completion time and financial cost of the queries, returning a priced contract to the consumer. If the consumer accepts the contract, in steps 4 through 6, the agent executes a job in the cloud according to the contract, computes the result, and returns a link to the consumer. The link can be, for example, an URL pointing to Amazon S3 or any other cloud storage service. Finally the agent receives payment based on the accepted contract and the actual completion time. We will see in Section 3.2 that contracts can involve complex prices that depend on the actual completion time.

2.2 Contracts

The contract is the core component of our framework, describing the terms of a computational task the agents will perform and the price they will receive upon task completion. The design of our framework is intended to cope with the inevitable uncertainty of completion time. Therefore, our contracts support variable pricing based on the actual completion time when the answer is delivered.

We also formally model the time/cost preferences of the consumer using a *utility function* that we assume is shared with the agent. The main technical challenge for the agent is to price a contract of interest to a consumer. Pricing relies on the agent's model of expected completion time for the task and the consumer's utility. From the consumers' side, they may receive and compare contracts from multiple agents to choose the one that maximizes their utility.

In this paper, we consider contracts and computational tasks that only involve analytic workloads. These analytic workloads are different from long-running services in the sense that their evaluation takes limited amount of time, even though this time can be several hours or days. Given this focus, we can assume that cloud resources remain the same during the execution of a contract. We discuss relaxing these factors in Section 8.

2.3 Properties and assumptions

Our framework is designed to support 3 important properties: competitiveness, fairness, and resilience. *Competitiveness* guarantees that agents have an incentive to reduce runtime and/or cost for consumers. *Fairness* guarantees that agents have an incentive to present accurate estimates to consumers, and that they do not benefit by lying about expected completion times. *Resilience* means that an agent can profit even when their estimates of completion time are imprecise and possibly erroneous. We demonstrate empirically in Section 6 that our framework satisfies these crucial properties.

Our framework assumes honest participants: we defer the study of malicious consumers and agents to future work. Accepting an agent's contract means the consumer's data will be shared with the agent for evaluation of their task, however requesting contract prices from a set of agents reveals only the consumer's statistics and task description.

Monopoly is not possible in this framework, and collusion among agents is unlikely.[3] First, an agent cannot constitute a monopoly, as consumers may always choose to use a cloud service provider directly. A service provider cannot be a monopoly either, as any agent with a valid estimation model can enter the market. Second, collusion becomes unlikely as the number of agents in the market increases. Any agent who does not collude with others can offer a lower price and draw consumers, making any collusion unstable.

3. THE CONSUMER'S POINT OF VIEW

In this section, we describe the consumer's interactions with the market. A transaction begins with a consumer who submits a request reflecting their *utility* (a precise description of their preferences). Later, given multiple priced contracts, the consumers can formally evaluate them according to the likely utility they will offer.

3.1 Consumer utility

One of our goals is to avoid simplistic definitions of contracts in which a task is carried out by a deadline for a single price. Many consumers have preferences far more complex than individual deadlines: they can tolerate a range of completion times, assuming they are priced appropriately. Also, we want agents to compete to offer contracts that best meet the preferences of consumers.

A consumer's preferences are somewhat complex because they involve tradeoffs between both completion time and price. We adopt the standard economic notion of consumer *utility* [40] and model it explicitly in our framework. A utility function precisely describes a consumer's preferences by associating a utility value with every (time, price) pair. A utility function can encode, e.g., the fact that the consumer is indifferent to receiving their query answer in 10 minutes at a cost of \$2.30 or 20 minutes at a cost of \$1.90 (when these two cases have equal utility values) or that receiving an answer in 30 minutes at a cost of \$0.75 is preferable to both of the above (when it has strictly greater utility).

DEFINITION 3.1 (UTILITY). Utility $U(t,\pi)$ *is a real-valued function of time and price, which measures consumer satisfaction when a task is evaluated in time t with price* π.

Larger values for $U(t,\pi)$ mean greater utility and a preferred setting of t and π. For a fixed completion time t_0, a consumer always prefers a lower price, so $U(t_0,\pi)$ increases as π decreases. Similarly, for a fixed price, π_0, a consumer always prefers a shorter completion time, so $U(t,\pi_0)$ increases with decreasing t.

To simplify the representation of a consumer's utility, we will restrict our attention to utility functions that are piecewise linear. That is, we assume the range of completion times $[0,\infty)$ is divided into a fixed set of intervals, and that utility on each interval is defined by a linear function of t and π. This means that for each interval, the consumer has a (potentially different) rate at which she/he is willing to trade more time for lower price, and vice versa.

DEFINITION 3.2 (UTILITY – PIECEWISE). *A piece-wise utility function consists of a list of target times* $\tau_0,\dots,\tau_n$, *where* $0 = \tau_0 < \tau_1 < \cdots < \tau_{n-1} < \tau_n = \infty$, *and linear functions* $u_1(\pi,t),\dots,u_n(\pi,t)$. *The utility is* $u_i(\pi,t)$ *for* $t \in [\tau_{i-1},\tau_i)$.

Such utility functions can express conventional deadlines, but also much more subtle preferences concerning the completion time and price of a computation. In practice, users can construct the utility function by defining several critical points on a graphical user interface, or answering a few simple pair-wise preference questions.

[3] In fact, the agents and the existing cloud service providers naturally form a monopolistic competition [40].

EXAMPLE 3.3. *Consumer Carol has two target completion times for her computation: 10 minutes and 20 minutes. Results returned in less than 10 minutes are welcome, but she doesn't wish to pay more to speed up the task. When results are returned between 10 minutes and 20 minutes, every minute saved is worth 1 cent to her. She does not want result returned after 20 minutes. Her piecewise utility function is:*

$$U(t,\pi) = \begin{cases} u_1(t,\pi) = -\pi & (t < 10) \\ u_2(t,\pi) = -t-\pi+10 & (10 \le t < 20) \\ u_3(t,\pi) = -50 & (t \ge 20) \end{cases}$$

3.2 Consumer contract proposal

The process of agreeing on a contract starts with the consumer advertising to agents the basic terms of a contract: the task $\mathbf{Q}$, the statistics of the database s_D, and their piecewise utility function U.

The terms of the contract are structured around the target times in the utility function. Agents use the utility function to choose a suitable configuration and pricing to match the preferences of the consumer. A complete, priced contract is returned to the consumer, which is defined as follows:

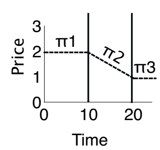
DEFINITION 3.4 (CONTRACT). *A contract is a six-tuple $\mathscr{C} = (\mathbf{Q}, s_D, \mathscr{T}, P, \hat{T}, \Pi)$, where $\mathbf{Q}$ is a task, s_D consists of statistics about the input data, $\mathscr{T} = (\tau_0, \tau_1, \ldots, \tau_n)$ is an ordered list of target completion times, $P = (p_1, \ldots, p_n)$ is an ordered list of probabilities, $\sum_i p_i = 1$, $\hat{T} = (\hat{t}_1, \ldots, \hat{t}_n)$ is an ordered list of expected completion times, and $\Pi = (\pi_1(t), \ldots, \pi_n(t))$ is a list of price functions where π_i is defined on $[\tau_{i-1}, \tau_i)$.*

When a consumer and an agent agree on a contract $\mathscr{C}$, it means that the agent has promised to deliver the answer to task $\mathbf{Q}$ on D after time $t \in [0,\infty)$, where the likelihood that t falls in interval $[\tau_{i-1}, \tau_i)$ is p_i. If the answer is delivered in interval $[\tau_{i-1}, \tau_i)$, the consumer agrees to pay the specified price, $\pi_i(t)$. $\hat{T}$ is used for computing expected utility as we will see in Section 3.3. The contract also includes the data statistics s_D, given to the agent by the consumer, because the pricing calculation relies on these statistics.

The contract is an agreement to run the task once. The probabilities provided by the agent are a claim that if the task were run many times, a fraction of roughly p_i of the time, the completion time would be in the interval $[\tau_{i-1}, \tau_i)$. Without this information, the consumer has no way to effectively evaluate the alternative completion times that could occur in a contract. For example, all alternatives but one could be very unlikely and this would change the meaning of the contract. We will see in Section 4 how the agent generates these probabilities.

EXAMPLE 3.5. *An example contract based on the utility function of Example 3.3 is defined by $\mathscr{T} = (0, 10, 20, \infty)$, probabilities $P = (0.2, 0.5, 0.3)$, expectations $\hat{T} = (9, 15, 21)$, and prices Π:*

$$\Pi(t) = \begin{cases} \pi_1(t) = 2 & (t < 10) \\ \pi_2(t) = 3 - 0.1t & (10 \le t < 20) \\ \pi_3(t) = 1 & (t \ge 20) \end{cases}$$



3.3 Consumer's contract evaluation

In response to a proposed contract, a consumer hopes to receive a number of priced versions of the contract from agents. Contracts based on the same utility request will share the same target completion times, but each contract may offer the consumer different utility values over the probability-weighted completion times. The consumer's goal is to maximize their utility, so to choose between contracts, the consumer should compute the expected utility of each contract and choose the one with greatest expected utility.

DEFINITION 3.6 (EXPECTED UTILITY OF A CONTRACT). *The expected utility of a contract $\mathscr{C} = (\mathbf{Q}, s_D, \mathscr{T}, P, \hat{T}, \Pi)$ with respect to utility function $U(t,\pi)$ is*

$$\textstyle\sum_{i=1}^{n} p_i u_i(\hat{t}_i, \pi_i(\hat{t}_i))$$

when $u_i(t,\pi)$ and $\pi_i(t)$ are linear functions.

EXAMPLE 3.7. *Suppose the consumer uses the utility function in Example 3.3, and two agents return two contracts $\mathscr{C}_1$ and $\mathscr{C}_2$. Further assume both agents return the same price function Π in Example 3.5, and the expected time $\hat{T}$ are also the same ($\hat{T} = (9, 15, 21)$). Only the probabilities P differ: $P = (0.2, 0.5, 0.3)$ for contract $\mathscr{C}_\mathbf{1}$, and $P = (0.1, 0.8, 0.1)$ for contract $\mathscr{C}_\mathbf{2}$. The consumer computes the expected utility according to Definition 3.6 (-18.65 for $\mathscr{C}_\mathbf{1}$ and -10.4 for $\mathscr{C}_\mathbf{2}$), and chooses $\mathscr{C}_2$ as it has greater utility.*

4. THE AGENT'S POINT OF VIEW

We now explain the agent's interactions in the market. The agent's challenge is to price a contract, coping with the uncertainty of completion time, while considering the consumer's utility and the market demand. We formalize 2 variants of pricing (risk-aware and risk-agnostic) and formulate both as optimization problems.

4.1 Pricing preliminaries

Upon receipt of the terms of a contract and the utility function of a consumer, the agent must complete the contract by computing prices for each interval and assigning probabilities to each interval.

For each configuration, we assume the financial cost C borne by the agent is a function of t: $C(t) = \alpha_C \cdot t$, where α_C is the unit rate of the configuration, and can be different across configurations. Thus, the pricing of a contract depends critically on the estimate of the completion time for $\mathbf{Q}$. Since estimates of completion time are uncertain, we model completion time T as a probability distribution over $[0,\infty)$ with probability density function $f_T(t)$. The true $f_T(t)$ is unlikely to be known and, in practice, must be estimated by the agent with respect to a selected configuration. Based on $f_T(t)$ and $C(t)$, the agent proposes a price function $\pi(t)$, which means the consumer should pay $\pi(t)$ when the completion time is t.

The agent has three goals when pricing a contract: (i) to maintain profitability, (ii) to offer the consumer appealing utility, and (iii) to compete with the offerings of other agents. We discuss them below.

(i) Profitability.

Naturally the agents would like to price the contract higher than their cost of execution so that they can earn a profit. Profit is uncertain for an agent because it is difficult to predict completion time in the cloud. We say a contract is *profitable in expectation* if its expected profit, with respect to the distribution $f_T(t)$, is greater than 0.

$$E[profit] = \textstyle\sum_{i=1}^{n} p_i\,(\pi_i(\hat{t}_i) - C(\hat{t}_i)) \tag{4.1}$$

We call a contract profitable (for the agent) as long as it is profitable in expectation. The agents should always price contracts so that they are profitable, but it is possible that a particular contract ends up being unprofitable.

DEFINITION 4.1 (PROFITABLE CONTRACT). *A profitable contract is a contract with $E[profit] > 0$.*

(ii) Prioritizing consumer utility.
Since the agents know the consumer's utility function $U(t,\pi)$ they should take it into account when choosing a configuration and pricing. To the extent that the agents can match the consumer's utility, their pricing of the contract will be more appealing to the consumer. The agents evaluate the expected utility $E[U]$ over the distribution of time T based on their estimates and price function $\pi(t)$:

$$E[U] = \sum_{i=1}^{n} p_i u_i(\hat{t}_i, \pi_i(\hat{t}_i)) \tag{4.2}$$

Profitability for the agent and utility for the consumer are conflicting objectives: a contract that offers greater profit to the agent will offer lower utility to the consumer. We will see that the agent will attempt to maximize the consumer's utility, subject to constraints on their profitability.

(iii) Market competitiveness and demand.
In all markets, including ours, market forces and competition prevent agents from raising prices without bound. In economics, a market demand function describes how these forces impact the pricing of goods [40]. When the agents decrease the price of a contract, the expected profit of the contract is reduced but they increase the utility of the contract to consumers. In a marketplace, when utility for the consumer increases, a greater number of consumers will accept the contract. Thus, the agents must balance the profit made from an individual contract with the overall profit they make from selling more contracts. To model this, we must make an assumption about the relationship between utility and the number of contracts that will be accepted by consumers in the market. This relationship is represented by the *demand function* which is defined as a function of utility. A linear demand curve is common in practice [40], so we focus on demand functions of the form $M(U) = a + bU$. Our framework can support demand functions of different forms, but we do not discuss these in detail.

In a real market, agents would learn about demand through repeated interactions with consumers. An agent's demand function could depend on, for example, customer loyalty, the best contracts competitors can offer, and other factors. These factors are beyond our scope. To simulate the functioning of a realistic market, we must assume a demand function and, for simplicity, we assume the demand functions of all agents are the same in the rest of this paper.

4.2 Contract pricing

We start from the simplest case in which the consumer has a task $\mathbf{Q}$ and a single configuration ϕ. So the cost function $C(t)$ and the pdf of the distribution of completion time $f_T(t)$ are fixed. The agent needs to define the price function $\pi(t)$ to present a competitive contract to the consumer. Let the overall profit be $\mathscr{P}$, which equals the unit profit *profit* multiplied by the sales $M(U)$. Notice that *profit* is the profit of a single contract while $\mathscr{P}$ is the overall profit of all contracts that the agent returns to all consumers in the market. The agent wants to find the price function that leads to the greatest total profit while satisfying the profitability constraint. This results in the following optimization problem:

PROBLEM 4.2 (CONTRACT PRICING). *Given a contract $\mathscr{C} = (\mathbf{Q}, s_D, \mathscr{T}, P, \hat{T}, \Pi)$, utility function U, and demand function M, the optimal price for $\mathscr{C}$ is:*

$$\begin{aligned} \textit{maximize}: &\quad \mathscr{P} = E[\textit{profit}] \cdot E[M(U)] \\ \textit{subject to}: &\quad E[\textit{profit}] > 0 \end{aligned}$$

Let I_i be the interval (t_i, t_{i+1}), and recall that p_i is the probability that the completion time falls in I_i:

$$p_i = \int_{t=t_i}^{t_{i+1}} f_T(t)dt \tag{4.3}$$

Let T_i be a random variable of completion time in interval I_i. It is a truncated distribution with probability density function $f_T(t|t \in I_i)$. Let C_i be a random variable of cost in interval I_i. $C_i = C(T_i)$. So expectation $\hat{t}_i$ and expectation c_i is:

$$\hat{t}_i = E[T_i] = \int_{t \in I_i} t f_T(t|t \in I_i)dt \tag{4.4}$$

$$c_i = E[C_i] = \int_{t \in I_i} C(t) f_T(t|t \in I_i)dt \tag{4.5}$$

Therefore the expected unit profit and expected demand are:

$$E[\textit{profit}] = \sum_{i=1}^{|I|} (\pi_i - c_i) p_i \tag{4.6}$$

$$E[M(U)] = \sum_{i=1}^{|I|} M(U(\hat{t}_i, \pi_i)) p_i \tag{4.7}$$

Linear case.
When U and M are linear functions, this problem becomes a convex quadratic programming problem. It has an analytical solution. More details can be found in the technical report [42]. Here we describe the conclusion only, under the following assumptions:

- The consumer specifies a linear utility function $U(t,\pi) = -\alpha_U \cdot t - \beta_U \cdot \pi$, which means they are always willing to pay α_U units of cost to save β_U units of time.
- The demand function is linear: $M(U) = \gamma_M + \lambda_M \cdot U$. Thus, when U increased by $1/\lambda_M$, 1 more contract would be accepted. Since $U(t,\pi)$ is linear, the demand function can be written as $M(U) = \gamma_M - \alpha_M t - \beta_M \pi$.

Applying Equations 4.6 and 4.7 to Problem 4.2, we compute the overall profit $\mathscr{P}$. $\mathscr{P}$ is maximized when

$$\pi^T \mathbf{p} = \begin{cases} (\gamma_M - \alpha_M \hat{\mathbf{t}}^T \mathbf{p} + \beta_M \mathbf{c}^T \mathbf{p})/(2\beta_M), & \gamma_M - \alpha_M \hat{\mathbf{t}}^T \mathbf{p} - \beta_M \mathbf{c}^T \mathbf{p} \geq 0 \\ \mathbf{c}^T \mathbf{p} + \varepsilon, & \textit{otherwise} \end{cases}$$

where ε is a small positive value, $\pi = [\pi_1, \pi_2, ...]^T$, $\mathbf{p} = [p_1, p_2, ...]^T$, $\hat{\mathbf{t}} = [\hat{t}_1, \hat{t}_2, ...]^T$ and $\mathbf{c} = [c_1, c_2, ...]^T$.

Selecting a configuration.
An agent typically has many available configurations for evaluating $\mathbf{Q}$. We denote the set of configurations by $\Phi = \{\phi_1, \phi_2, ...\}$. Every configuration ϕ_j has its own cost function $C_j(t) = \alpha_{Cj} \cdot t$, where α_{Cj} is the unit rate for ϕ_j. The agent will select the configuration that results in the most profit. The distribution of time T and its corresponding p_i, $\hat{t}_i$, and c_i then become variables in Problem 4.2. Each agent may use a different strategy to derive a solution. A naïve agent can select and enumerate a small Φ, while a smarter agent will use an analytic model to solve the problem [45, 16].

4.3 Risk aware pricing

Pricing contracts involves some risk for the agents: if their estimated distributions of time and cost are different from the actual ones, they can lose profit or even suffer losses. Next, we formally define risk based on loss and add it as part of the objective.

DEFINITION 4.3 (LOSS). *Let the actual distribution of completion time be T^* and the optimal price function be π^*. When the agent generates a contract with price function π, the loss of revenue L is: $L_{T^*}(\pi) = \mathscr{P}(\pi^*, T^*) - \mathscr{P}(\pi, T^*) = \mathscr{P}(\pi^*, p^*, t^*, c^*) - \mathscr{P}(\pi, p^*, t^*, c^*)$, where p^* is the actual probabilities, t^* is the actual expected completion times, and c^* is the actual costs.*

There is always inherent uncertainty in the prediction of the distributions of completion time and cost, so it is generally not possible for the agents to achieve the theoretically optimal profits based on the actual distributions. However, they can plan for this risk, and assess how much risk they are willing to assume. We proceed to define risk as the worst-case possible loss that an agent can suffer.

DEFINITION 4.4 (RISK). *The risk of the agent is a function of price π, and is defined as the maximum loss over possible distributions of completion time:* $R(\pi) = \max_{T^*} L_{T^*}(\pi)$.

We incorporate risk into the agent's optimization problem by adding it to the objective function:

$$\begin{aligned} &\text{maximize} : \mathscr{P}(\pi, p, t, c) - \lambda R(\pi) \\ &\text{subject to} : E[\mathit{profit}] > 0 \end{aligned} \tag{4.8}$$

The λ in the objective is a parameter of risk that the agent is willing to assume. Larger λ reduce the worst-case losses (conservative agent), while smaller λ increase the assumed risk (aggressive agent). The agent can estimate the risk $R(\pi)$ by solving the following optimization problem, with variables π^*, p^*, t^*, and c^*:

$$\begin{aligned} \text{maximize} : \; & L_{T^*}(\pi) = \mathscr{P}(\pi^*, p^*, t^*, c^*) - \mathscr{P}(\pi, p^*, t^*, c^*) \\ \text{subject to} : \; & E[profit^*] = \sum(\pi_i^* - c_i^*)p_i^* > 0 \\ & LBound_t \leq t_i^* - \hat{t}_i \leq UBound_t \\ & LBound_c \leq c_i^* - c_i \leq UBound_c \\ & 0 \leq p_i^* \leq 1, \; \sum p_i^* = 1 \end{aligned}$$

where *LBound* and *UBound* are empirical values set by the agent. For example, if the agent finds after several contract executions that the mean time is 1 sec higher than the estimate produced by the agent's analytic model, the agent can set $LBound_t = 0$ and $UBound_t = 1$.

5. FINE GRAINED CONTRACT PRICING

Our treatment of pricing in Section 4 assumes that agents select a single configuration for the execution of a consumer contract. However, computational tasks often contain well-separated, distinct subtasks (e.g., operators in a query plan or components in a workflow). These subtasks may have vastly different resource needs. For example, Juve et al. [19] profile multiple scientific workflow systems and find that their components have dramatically different I/O, memory, and computational requirements.

We now extend our pricing framework to support *fine-grained* pricing, which allows agents to optimally assign separate configurations to each subtask of a computational task. It provides more candidate contracts without changing the pricing Problem 4.2. Fine-grained pricing has two benefits. First, by assigning a configuration for each subtask, instead of the entire task, agents can achieve improved time and cost, resulting in higher overall utility and/or higher profit. Second, considering subtasks separately gives agents the flexibility to outsource some computation to other agents. While outsourcing computation across agents is not a focus of our work, it is a natural fit for our fine-grained pricing mechanism. Agents can choose to outsource subtasks to other agents based on their specialization and capabilities, or for load balancing. However, some challenges of outsourcing, such as utility and forms of contracts that agents need to exchange are beyond the scope of our current work.

We model a computational task $\mathbf{Q}$ as a directed acyclic graph (DAG) $G_\mathbf{Q}$. Every node in $G_\mathbf{Q}$ is a subtask Q_i. An edge between subtasks (Q_i, Q_j) means that the output of Q_i is an input to Q_j. When subtasks are independent of one another, the DAG may be

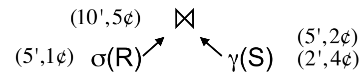


Figure 2: An example of a simple relational query that can be broken into 3 subtasks, corresponding to different operators.

disconnected. Our model assumes no pipelining in subtask evaluation. Therefore, a subtask Q_j cannot be evaluated until all subtasks Q_i, such that $(Q_i, Q_j) \in G_\mathbf{Q}$, have completed their execution.

Given the graph representation $G_\mathbf{Q}$ of a computational task, an agent determines a configuration $\phi_i \in \Phi$ for each subtask $Q_i \in \mathbf{Q}$. This is in contrast with coarse-grained pricing (Section 4.2), where the agent selects a single configuration from Φ for all subtasks of $\mathbf{Q}$. When the agent chooses ϕ_i, the time and cost of Q_i is $T_i(\phi_i)$ and $C_i(\phi_i)$. A set of selected configurations results in total cost $C_\mathbf{Q} = \sum_{Q_i} C_i(\phi_i)$, i.e., the sum of the costs of all subtasks. The completion time of $\mathbf{Q}$ is determined by the *longest* path (P) in the task graph: $T_\mathbf{Q} = \max_{P \in G_\mathbf{Q}} \sum_{Q_i \in P} T_i(\phi_i)$. Given demand M and contract utility U, $T_\mathbf{Q}$ and $C_\mathbf{Q}$ determine the agent's profit $\mathscr{P}$. The goal of the agent is to select the set of configurations that maximizes $\mathscr{P}$.

PROBLEM 5.1 (FINE-GRAINED CONTRACT PRICING). *Given graph $G_\mathbf{Q}$ representing a task $\mathbf{Q}$, and possible configurations Φ, the agent needs to specify a configuration $\phi_i \in \Phi$ for each $Q_i \in \mathbf{Q}$, so that the time $T_\mathbf{Q} = \max_{P \in G_\mathbf{Q}} \sum_{Q_i \in P} T_i(\phi_i)$ and cost $C_\mathbf{Q} = \sum_{Q_i} C_i(\phi_i)$ maximize the overall profit $\mathscr{P}$.*

Our problem definition does not model data storage and transfer time and costs explicitly. Rather, we assume that these are incorporated in the time and cost of a subtask (T_{Q_i} and C_{Q_i}). This simplifies the model and offers an upper bound on time and cost. In practice, when two subsequent tasks share the same configuration, it is possible to reduce the costs of data passing, but these optimizations are beyond the scope of this work.

We demonstrate the intricacies of the fine-grained pricing problem through a simple example. Figure 2 shows a relational query with three distinct subtasks (operators): (1) select tuples from relation R, (2) aggregate on relation S, and (3) join of the results. We assume deterministic times and costs to evaluate each subtask, denoted next to each node in Figure 2. The select and join subtasks have only a single possible configuration each, but the aggregate subtask has two. Assume the utility function is $U(t, \pi) = -t - \pi$, which means every one minute is worth 1 cent for the consumer. Therefore, the configuration $(2', 4¢)$ is better for the aggregate subtask, since it has higher utility than the configuration $(5', 2¢)$. However, following a greedy strategy that picks the configuration that is optimal for each subtask can result in sub-optimal utility for the overall task. In this example, the join subtask has to wait 5 minutes for the select subtask to complete. Therefore, there is no benefit in paying a higher price to complete the aggregate subtask sooner, making $(5', 2¢)$ a better configuration choice.

THEOREM 5.2. *Fine-Grained Contract Pricing is NP-hard.*

Our reduction follows from the discrete Knapsack problem [42]. We next introduce a pseudo-polynomial dynamic programming algorithm for this problem, and show that it is efficient and effective in real-world task workflows (Section 6.3). Without loss of generality, we assume time and cost are deterministic, but the algorithm can be extended to the probabilistic case in a straightforward way.

Algorithm 1 uses dynamic programming to compute the optimal profit for task graphs. The algorithm derives the exact optimal solution for cases where $G_\mathbf{Q}$ is a tree (e.g., relational query operators) and computes an approximation of the optimum for DAGs.

Algorithm 1 Fine-Grained Contract Pricing

Require: $\mathbf{Q}, G_{\mathbf{Q}}, \Phi, \mathscr{P}(T,C)$
Ensure: *maximum* $\mathscr{P}$
1: Add node $Q_{terminal}$ with 0 time and cost to $G_{\mathbf{Q}}$
2: **for all** $Q_i \in \mathbf{Q}$ **do**
3: Add edge $(Q_i, Q_{terminal})$ to $G_{\mathbf{Q}}$
4: $\mathbf{Q}_{order} \Leftarrow TopologicalSort(G_{\mathbf{Q}})$
5: $boundT \Leftarrow$ longest time to evaluate $G_{\mathbf{Q}}$
6: **for all** $Q_i \in \mathbf{Q}_{order}$ **do**
7: $f(Q_i, 0) \Leftarrow \infty$
8: **for** $t \Leftarrow 1$ **to** $boundT$ **do**
9: $f(Q_i, t) \Leftarrow f(Q_i, t-1)$
10: **for all** $\phi \in \Phi$ **do**
11: $cost_\phi \Leftarrow \underset{q \in pred(Q_i)}{Combine}(f(q, t - T_i(\phi))) + C_i(\phi)$
12: **if** $cost_\phi < f(Q_i, t)$ **then**
13: $f(Q_i, t) \Leftarrow cost_\phi$
14: **return** $\max_t \mathscr{P}(t, f(Q_{terminal}, t))$

In Algorithm 1, $f(Q_i, t)$ represents the minimum cost for evaluating the subgraph terminated at subtask Q_i when it takes at most time t.[4] Then, $f(Q_i, t)$ can be computed based on a combination of the costs of the direct predecessors of Q_i ($pred(Q_i)$) in the workflow (lines 7–13). When $G_{\mathbf{Q}}$ is a tree, the *Combine* function (line 11) is simply the sum of the costs of the predecessors ($\sum_{q \in pred(Q_i)} f(q, t - T_i(\phi))$), and Algorithm 1 results in the optimal profit.

If $G_{\mathbf{Q}}$ is not a tree, predecessors of a subtask Q_i can share common indirect predecessors, which introduces complex dependencies in the choice of configurations across different subtrees. For example, let $q_1, q_2 \in pred(Q_i)$, and $q_0 \in pred(q_1) \cap pred(q_2)$. Therefore, q_0 affects both subgraphs terminated at q_1 and q_2, respectively. This impacts the *Combine* function in two ways. First, the cost of q_0 should be counted only once. Second, there may be discrepancies in the configuration choice for q_0 by the different subgraphs. There are three strategies to resolve the discrepancy: (1) use the configurations with minimum time T; (2) use the configurations with minimum cost C; (3) use the configurations with maximum $\mathscr{P}(T,C)$. The *Combine* function applies the above strategies one by one, computes the time T_{Q_i} and cost C_{Q_i} of the subgraph terminated at Q_i, and updates $f(Q_i, t)$ if $T_{Q_i} \leq t$ and C_{Q_i} is better. Note that Strategy 1 guarantees a feasible solution whenever one exists.

6. EXPERIMENTAL EVALUATION

In this section we evaluate our market using a real-world cloud computing platform: Amazon Web Services (AWS). Our experiments collect real-world data from a variety of relational and MapReduce task workloads, and use this data to simulate the behavior of our market entities on the AWS cloud. Our results demonstrate that our market framework offers incentives to consumers, who can execute their tasks more cost-effectively, and to agents, who make profit from providing fair and competitive contracts.

We proceed to describe our experimental setup, including computational tasks, consumer parameters, and contracts.

Data and configurations.

We spent 8,106 machine hours and $3,118 in obtaining the distributions of time and cost for two types of computational tasks: relational query workloads, and MapReduce jobs.

Relational query tasks: We use the queries and data of the TPC-H benchmark to evaluate relational query workloads. We use all 22 queries of the benchmark on a 5GB dataset (scale factor 5). We

[4]We turn the continuous space of time t into discrete space by choosing an appropriate granularity (e.g., minute).

Type	CPU (virtual)	Memory	$/hour
db.m3.Medium	1	3.75GB	$0.095
db.m3.Large	2	7.5GB	$0.195
db.m3.xLarge	4	15GB	$0.390
db.m3.2xLarge	8	30GB	$0.775
db.r3.Large	2	15GB	$0.250
db.r3.xLarge	4	30.5GB	$0.500
db.r3.2xLarge	8	61GB	$0.995
m1.Medium	1	3.75GB	$0.109
m1.Large	2	7.5GB	$0.219
m1.xLarge	4	15GB	$0.438

Figure 3: Types of Amazon machines and associated features and costs (in January 2015). The first 7 types (db.*) are RDS configurations, whereas the last 3 (m1.*) are EMR configurations. The prefixes (db and m1) are omitted from some figures for brevity.

use the Amazon Relational Database Service (RDS) to evaluate the workloads on 7 machine configurations, each of which has 200GB of Provisioned IOPS SSD storage, and runs PostgreSQL 9.3.5. Figure 3 lists the capacity and hourly rate of each configuration.

MapReduce tasks: We evaluate MapReduce workloads using 3 job types (WordCount, Sort, and Join) over 5GB of randomly generated input data. We use the Amazon Elastic MapReduce service (EMR) to test our framework. We select 3 machine configuration types. Figure 3 lists the capacities and hourly rates of these configurations. We experimented with 4 different sizes of clusters for each machine configuration: 1, 5, 10, and 20 slave nodes.

Scientific workflows: We use real-world scientific workflows that represent computational tasks with multiple subtasks, to evaluate fine-grained pricing (Section 5). We retrieved 1,454 workflows from MyExperiment [10], one of the most popular scientific workflow repositories. These workflows were developed using Taverna 2 [44], and comprise the majority of workflows in the repository. The size of workflows ranges from 1 to 154 subtasks.

Consumer models.

We simulate the consumer behavior in our framework using the utility and demand functions.

Utility: In our evaluation, utility is a linear function $U(t,\pi) = -\alpha_U t - \beta_U \pi$ modeling consumer preferences. α_U represents the unit cost that the consumer is willing to pay to save β_U unit time. For our experiments, we assume $U(t,\pi) = -t - \pi$, where t is measured in minutes and π is measured in cents, which means every minute is worth 1 cent to the consumer.

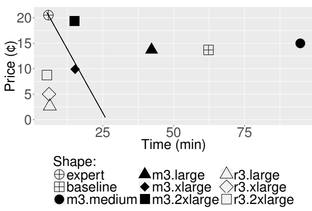
Demand: In our evaluation, the demand function is linear: $M(U) = \gamma_M + \lambda_M U$, which means that when U increases by $1/\lambda_M$, 1 more contract would be accepted. We use $M(U) = 100 + 50U$ for RDS, and $M(U) = 100 + 5U$ for EMR. λ_M is smaller for EMR because the times and costs for MapReduce jobs are much larger than those of relational queries.

Contracts.

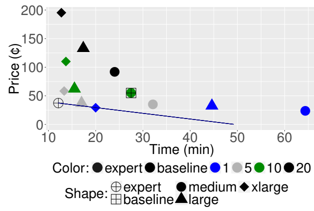
All our experiments involve contracts with a deadline, which means that every consumer request specifies one target completion time. We execute each task 100 times using every configuration and set the deadline of each query as the average completion time across all configurations.

6.1 Consumer incentives

In this section, we evaluate whether our market framework offers sufficient incentive for consumers to participate in the market. Our first set of experiments simulates several naïve cloud users who select one of the default configurations for their computational tasks:

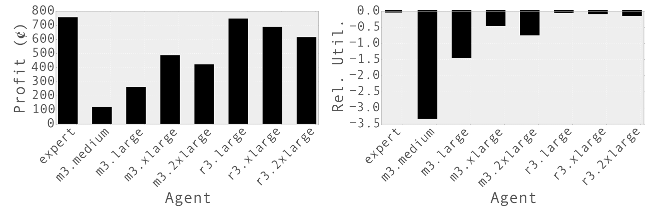


(a) Utility trade-off (RDS)

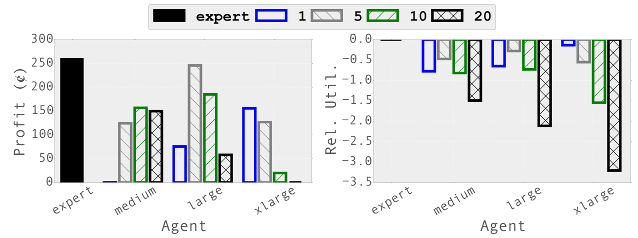


(b) Utility trade-off (EMR)

Figure 4: Users achieve better utility by using an expert agent, compared to naïvely selecting a default configuration. The agent benefits 40% of the consumers in RDS workloads, and 100% of the consumers in EMR workloads.



(a) Agent profits and relative utilities (RDS)



(b) Agent profits and relative utilities (EMR)

Figure 5: Expert agents always achieve the largest profits: our framework incentivizes agents to find optimal configurations.

7 configurations for RDS, and 12 configurations for EMR. We also simulate a baseline user who selects one of two cloud configurations, the one with the best CPU performance and the one with the best IO performance, based on the workload type: CPU-intensive or IO-intensive. To do this, we first executed each task to measure its CPU and IO time, and used this information to classify tasks as primarily CPU-intensive or primarily IO-intensive. This biases the experiment in favor of the baseline, as the consumer would not typically have such accurate classification in practice. Finally we simulate an expert agent, who, for every task, selects the configuration that maximizes the consumer's utility function. Figure 4a presents the price and time achieved by each of the 7 default configurations for RDS, as well as the price and time offered by the expert agent. The line in the graph is the utility indifference curve for the agent's configuration, representing points with the same utility value. Points on the curve are equally good, from the consumer's perspective, as the one achieved by the expert agent. Points above the curve have worse utility values (less preferable than the agent's offer), while points below the curve have better utility values (more preferable than the agent's offer).

Our experiments show that the expert agent provides more utility to 4/8 naïve cloud users with relational query tasks on RDS. This means that, even though the agent makes a profit, a good portion of the users still benefit from using the market instead of relying on default settings. This effect is even more pronounced for EMR workloads. Figure 4b shows that the expert agent offers better utility to *all* simulated naïve users. This means that, in every single case, the consumers get better utility by using the agent's services, instead of intuitively selecting a configuration. Notably, the baseline heuristic user also performs worse than the agent: it is 189% and 67% worse in utility for RDS and for EMR workloads, respectively.

6.2 Agent incentives and market properties

In this section we demonstrate that the pricing framework satisfies 3 important properties: *competitiveness*, *fairness*, and *resilience*. These properties incentivize consumers and agents to use and trust the market by ensuring that (a) the agents will identify efficient computation plans and provide accurate pricing, and (b) inaccurate estimates will not pose a great risk to the agents. A theoretical analysis of the market properties is in our technical report [42].

Competitiveness.

We run experiments on Amazon RDS and EMR to demonstrate how different configurations impact profitability in practice. Our goal is to show that, in our market, well-informed, expert agents can make more profit than naïve agents, thus incentivizing agents to be competitive and offer contracts that benefit the consumers. In this experiment, a naïve agent selects one configuration for all queries. In contrast, the expert agent always selects the optimal configuration for each query. The goal of this experiment is to show the impact of configuration selection. Thus we control for other parameters, such as the accuracy of the agents' estimates. So, for now, we assume that all agents know the distributions of time and cost accurately. We relax this assumption in later experiments.

We generate histograms of time and cost by evaluating each query with each configuration 100 times. All agents use these histograms to approximate the distributions and price contracts. After an agent prices a contract, we compute the number of accepted contracts according to the demand function, $M(U)$. Then we randomly select M executions to do trials. The agent receives payments based on whether the execution met or missed the deadline.

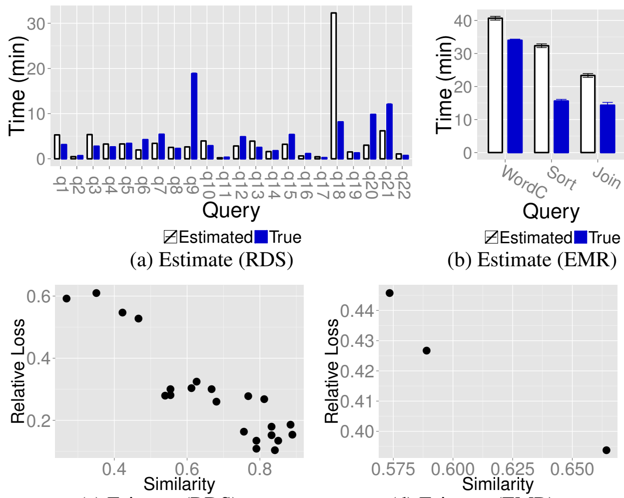
Figure 5a illustrates the total profit made by each agent pricing RDS workloads. There are 7 naïve agents, each using one of the RDS configurations from Figure 3, and one expert agent, who always uses the best configuration for each task. Figure 5b illustrates the same experiment on EMR workloads. We use one expert agent and 12 naïve agents who used the three EMR configurations from Figure 3, each with a cluster size of 1, 5, 10, or 20 nodes. The expert agent chooses 3 different configurations for the RDS tasks and 3 different configurations for the EMR tasks. The details are in our technical report [42]. In both experiments, the expert agent achieves the *highest* overall profit.

Figures 5a and 5b also show the utilities offered by the agents for the same contracts. We plot the relative utility of naïve agents, using the utility of the expert agent as a baseline: $\frac{AgentUtility - ExpertUtility}{|ExpertUtility|}$. On both RDS and EMR workloads, the utility offered by the expert was the best among all agents.

Our experiments on both RDS and EMR demonstrate that expert agents achieve better utility and profit than all other agents. This verifies empirically that our market design ensures incentives for agents to improve their estimation techniques and configuration selection mechanisms. This benefits both consumers, who get better utility, and agents, who get more profit.

Fairness.

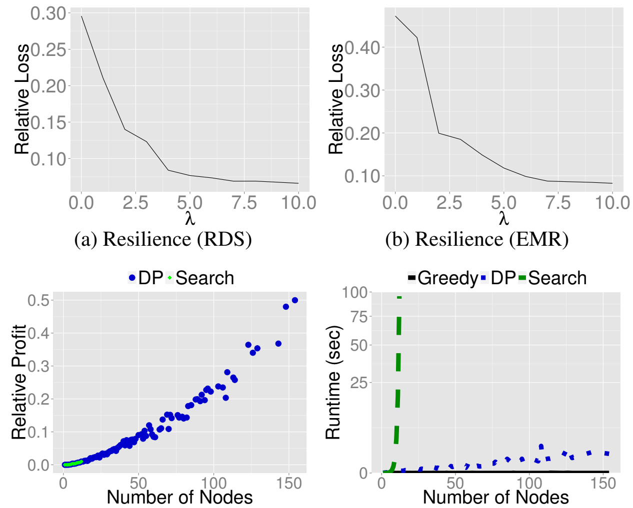
Fairness guarantees the incentive for agents to present accurate estimates to consumers. If the agent uses inaccurate estimates, she/he will be penalized with lower profits. Our goal is to show that more accurate estimates lead to greater profit for the agent in practice.



(a) Estimate (RDS) (b) Estimate (EMR)

(c) Fairness (RDS) (d) Fairness (EMR)

Figure 6: Agents' estimates are often inaccurate, and such inaccuracies can lead to loss of profit.



(a) Resilience (RDS) (b) Resilience (EMR)

(c) Profit Relative to Greedy (d) Runtime

Figure 7: (a, b) By adjusting for risk, agents can reduce their losses in case of inaccurate estimates. (c, d) DP outperforms Greedy and Search.

We consider an agent using db.m3.medium on RDS and PostgreSQL's default query optimizer to estimate the completion times of queries. The PostgreSQL optimizer provides an estimate of the expected completion time and the agent assumes a Gaussian distribution with a mean value equal to the completion time predicted by the optimizer. We chose 0.05 for the standard deviation, which is very close to the actual average standard deviation of the distributions of the 22 TPC-H queries (0.04).

We also consider another agent using m1.medium on EMR, with 1 master and 1 slave node. The agent estimates the expected completion time by executing queries on a 5% sample of the data, and assumes a Gaussian distribution around the estimated mean. The agent uses an empirical standard deviation, 0.55, which is close to the average true standard deviation of all 3 EMR job types (0.56).

We compare the agents' estimates with the true distributions in Figures 6a and 6b. We plot the average completion time for each TPC-H query and each EMR task. The standard deviation is very low (< 0.75 min) for all tasks. As these plots show, the agents' estimates can often be far from the actual completion times (e.g., q_{18}).

Next, we use the similarity between two distributions and relative loss to show the relationship between estimation accuracy and profit. We compare the true distribution of completion time (which is a histogram) with the agent's estimate (a Gaussian distribution) by turning the agent's estimate into a histogram and computing the cosine similarity between two histograms. The relative loss measures how much profit the agents lose compared to the optimal profit they could have made. We define relative loss of profit as:

$$RelativeLoss = \frac{OptimalProfit - ActualProfit}{OptimalProfit} \tag{6.1}$$

As Figures 6c and 6d illustrate, when the agent's estimate is more accurate, the relative loss is smaller.

These experiments show that our market does not rely on the assumption that the estimates are accurate, and it can tolerate inaccuracies well. If there exists at least one task for which an agent can produce better estimates than a consumer, the agent offers utility to the market. However, we also study an extreme case: when all agents in the market make worse estimation than all consumers, for all tasks. Overestimation leads agents to post higher prices lowering the consumers' utilities. However, agents have to overestimate time and cost by 49% in RDS workloads, and by 120% in EMR workloads before a switch is beneficial to consumers on average. On the other hand, underestimation of time and cost decreases agents' profits. Our evaluation shows that an agent will lose 2% profit in RDS and 36% in EMR if it underestimates time and cost by a factor of 10. Depending on the agents' profit margins, they may be able to absorb the difference without losses. To avoid losses, agents can follow risk-aware pricing strategies (Section 4.3). Details on this experiment and figures are in our technical report [42].

Resilience.

The property of *resilience* provides assurances to the agents, by ensuring that inaccurate estimates will not pose a significant risk to the agents' profits. This property is crucial, as errors in the estimates are very common [4, 11, 45]. Our framework ensures resilience to these inaccuracies by accounting for *risk* (Definition 4.4). Specifically, the agents can profit by adjusting the risk they prefer to take. According to Equation 4.8, the risk is part of the objective and controlled by a parameter λ. When λ is large, the agent has low confidence in the estimate (conservative). This setting reduces the loss of profit if the agent's estimate is inaccurate.

We again consider an RDS agent using db.m3.medium and the default PostgreSQL optimizer, and an EMR agent using m1.medium and sampling to estimate runtime. We evaluate relative loss using Equation 6.1 and plot it for different values of λ (Figure 7). A value of $\lambda = 0$ means that the agent is confident that their estimate is correct. However, since in this case the estimates were inaccurate, the relative loss for $\lambda = 0$ is high: the agents' profit is much lower than the optimal profit they could have achieved. For both agents (EMR and RDS), the relative loss decreases for higher values of λ. This shows that by adjusting for risk, the agents can reduce loss of profit.

6.3 Fine grained pricing

In our final set of experiments, we evaluate fine-grained pricing (Algorithm 1) against a large dataset of real-world scientific workflows [10]. This dataset is well-suited for this experiment, as it provides diverse computational flows of varied sizes and complexities. The published workflows do not report real execution information (time and cost), and we are not aware of any public workflow repositories that provide this information. Therefore, we augment the real workflow graphs with synthetic time and cost histograms for each subtask, drawn from random Gaussian distributions with means in the [1,100] range, and variances in the [0,5] range. Each subtask has 5 candidate configurations with different time and cost histograms. We compute the profit using utility $U(t,\pi) = -t-\pi$ and demand $M(U) = 100+0.01U$ (Section 4.2). We set λ_M (the coefficient of U) to a smaller value than the ones used for RDS and EMR workloads, because the completion times and costs for workflows are much larger.

First, we evaluate our Dynamic Programming algorithm (Algorithm 1) against two baselines: (1) an exhaustive search strategy (*Search*) that explores all possible configuration assignments, and (2) a greedy strategy (*Greedy*) that selects the configuration that leads to the maximum local profit for each subtask. We perform 10 repetitions for each workflow, using different random time and cost distributions for each repetition. Figure 7c shows the relative profit achieved by *Search* and *DP* compared to *Greedy*: $\frac{\mathscr{P}-\mathscr{P}_{Greedy}}{\mathscr{P}_{Greedy}}$. *DP* achieves better profits than *Greedy*, and the effect increases for larger workflows: for workflows with 154 subtasks, *DP* achieves 50.0% higher profit than *Greedy*. *Search* provides few data points, as it cannot scale to larger graphs. For small workflows (up to 12 subtasks) *Search* and *DP* select equivalent configurations that result in the same (optimal) profit. Figure 7d shows the running time of the three algorithms. As expected, exhaustive search quickly becomes infeasible, and *Greedy* is faster than *DP*. However, the runtime of *DP* remains low even for larger workflows. Combined with the profit gains over *Greedy*, this experiment demonstrates that Algorithm 1 is highly effective for fine-grained pricing.

Second, we evaluate the benefits of fine-grained pricing, compared to coarse-grained pricing. Figure 8a shows the profit achieved by *DP*, which assigns a configuration to each subtask, relative to the optimal single configuration for the entire workflow. In this experiment, fine-grained pricing doubled the agents' profits for small workflows, compared to coarse-grained pricing, and the gains increase as workflows grow larger. For the largest workflows in our dataset, fine-grained pricing achieved 12.5 times higher profits.

6.4 Comparison with Alternative Approaches

Benchmarking. Contrasting our work with Benchmark as a Service (BaaS) [12] is meaningful when workload repetition is significant. We assume a consumer who repeats RDS and EMR workloads without modifications, and with each repetition tests a different configuration; once all configurations are tested, the consumer uses the best one in subsequent repetitions. Clearly, after some number of repetitions, the benchmarking approach will outperform the agent-based approach. For this experiment, we limited the number of configurations to 7 for RDS and 12 for EMR. This biases the experiment in favor of benchmarking, as in practice the number of configurations that the consumer would have to try is much higher. Even in this simplified setting, we found that it took 12 repetitions in RDS and 68 repetitions in EMR before the consumer would start benefiting from benchmarking (Figure 8b). In the real world, these numbers are much higher, as cloud providers offer way more configurations than the ones we considered here. Cluster size alone causes an explosion in the number of options, so having an agent with an analytical model, such as in [16], is necessary.

In practice, BaaS has additional challenges. Data growth and changes in the input make BaaS complicated [12]. Workloads are almost never repeated exactly, as the input changes between executions, requiring the BaaS provider to monitor and react to changes. Moreover, cloud providers change machine types, parameters, and pricing very frequently — e.g., between 2012 and 2015, AWS introduced 2.6 new instances, on average, every three months. When these settings change, resource selection needs to be re-evaluated, even if a workload stays the same.

VCG auction. A Vickrey-Clarke-Groves (VCG) auction is a strategy-proof pricing mechanism. In this model, a customer opens a bidding and agents bid on prices. A VCG auction is strategy-proof, which means the agents truthfully reveal their best costs of executing a task. The consumer selects the agent with the best utility u_{1st} but pays according to the second best utility u_{2nd}. Therefore, given a specific task, only the best and the second best agents' contract determine the price. We simulate an agent who identifies the best configuration for each task, and another agent who is doing just worse than the first. We represent the utility difference with $\Delta = u_{1st} - u_{2nd}$, and vary Δ to determine how much profit the best agent can achieve.[5]

Figure 8c shows that initially agent profits grow with Δ, but eventually drops, due to the drop in demand. The maximum VCG profit is less than the profit in our agent-based market. Moreover, by the definition of VCG, the increase in Δ also causes an increase in the cost for consumers. Therefore, while agents make less profit in the VCG model, consumers do not get any benefits on average. Another challenge with the VCG model is that it requires a centralized auctioneer who ensures that consumers pay according to the second best utility. This makes a cross-platform market more difficult to form. Our agent-based model does not have such requirement.

7. RELATED WORK

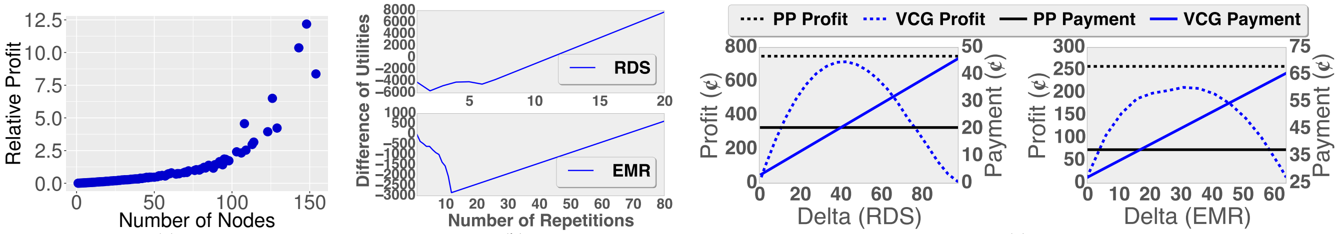
In contrast to our market framework, which emphasizes the consumer need for task-level pricing, existing work on cloud pricing largely focuses on resource usage. One study using game theory [5, 15] assumed that the price of resources impacts the demand and the quality of service (QoS), which in return affect the provider's revenue. This work makes two assumptions not present in our framework. First, their utility functions consider QoS degradation when consumers share resources. While meaningful for resources such as wireless bandwidth, this assumption does not always hold for computational resources [3]. In fact, QoS can improve when, e.g., consumers share data and cache. Second, they assume that the consumers know each other's demands and strategies, and adjust their demands accordingly. In contrast, we consider consumers' tasks separately and use probability distributions to model runtime and financial cost, leading to a simpler yet practical model.

Variants of pricing mechanisms assume that providers price dynamically, based on consumer arrival and departure rates [28, 47]. In turn, prices also guide consumer demand. In a different direction, Ibrahim et al. [17] argue that the interference across virtual machines sharing the same hardware leads to overcharging. They suggest that cloud providers price based on effective virtual machine time. This framework guarantees benefits to consumers and urges providers to improve their system design.

Wong et al. [18] compare 3 different pricing strategies in terms of fairness and revenue. Providers can lease separate resources (e.g., CPU, memory), a fixed bundle of resources (e.g., virtual machines), or personalized bundles of resources with differentiated prices. They treat consumers' jobs identically, define fairness based on the number of jobs executed by the provider, and conclude that differentiated pricing provides the best fairness. They do not consider the connection between uncertain completion time and utility.

Economic-based resource allocation has been extensively studied in grid computing [7, 32, 21]. Researchers develop economic

[5]The canonical VCG auction only models cost. We have extended the model to account for utility (cost and time), while preserving the strategy-proof property [42].



(a) (b) (c)

Figure 8: (a) Pricing at finer granularities vastly increases the agents' profits. (b) Consumers may prefer a benchmark-based approach for highly-repetitive workloads. (c) A VCG auction brings less profit to agents without necessarily reducing consumers' payments (PP = our posted-price approach).

models in two main categories: commodity markets and auctions. In a commodity market, resources are sold at a posted-price. The price of resources affects consumers' utility and demand, and therefore impacts the providers' profit. Yeo et al. [49] and Stuer et al. [36] apply differentiated pricing. Bossenbroek et al. [6] propose option contracts and use hedge strategies to reduce consumers' risk of missing deadlines. Auction-based pricing in grid computing has several different forms including double auction [37], Vickrey auction [29], and combinatorial auction [9].

Posted-price selling and auctions are both established market mechanisms, and one does not dominate the other. The key challenge is the uncertainty of the value of the commodity (in this case, the computational resource) and researchers develop different models to compare the two mechanisms under various assumptions. Computer scientists measure system metrics: posted-price brings more price stability, higher task completion rate, and higher resource utilization ratio than auctions [43] while Vickrey auction results in less message passing [39]. Economists have discussed the revenues: posted-price selling brings more profit to the seller when the buyers' values of the commodity are widely dispersed [41].

In contrast to our framework, this entire body of work focuses on resource-level pricing, and does not provide a mechanism for consumers to select resources based on their tasks. Recent work has started shifting the focus to task-level pricing. Benchmark as a Service (BaaS) [12] benchmarks users' workloads and suggests the optimal configuration for repetitive execution. Our experiments showed that consumers can benefit from benchmarking only when there is significant *exact* repetition. However, workloads are almost never repeated exactly, as the input data changes between executions. Moreover, cloud providers change their configuration offerings very frequently, which poses challenges to benchmarking in practice. Tanaka et al. [38] make providers bid for service contracts under the VCG auction. They do not consider execution time for tasks, while we balance the consumers' trade-off of time and price through their utility function. Ortiz et al. [30, 31] propose Personalized Service Level Agreements resembling the contracts in our framework, and describe a system that analyzes consumers' data and suggests to them tiers of service. A tier on AWS can be, for example, (< 3.5 minutes, \$0.12/hour, SELECT 1 attribute FROM 1 table WHERE condition). In our framework, consumers do not subscribe to a tier of service, but rather provide the task they need and the agent provides a specific price for the task. When multiple agents find the same best configuration for some tasks, their prices affect each other and, after several iterations, converge to the Nash Equilibrium in the differentiated Bertrand model [34].

The agents in our computation framework derive estimates of cost and time. Several approaches predict the runtime of a query using machine learning [13, 4], statistical models [23], sampling [11], or query plans [45]. Ye et al. [48] perform service composition given the resource requirement of individual tasks. Uncertainty of time and cost is an important component in our framework. Existing work on scheduling SLAs considers uncertainty in runtime when contracts specify a price. In such frameworks, agents receive payment when finishing tasks on time, or pay a penalty if they reject the task or miss the deadline. Researchers develop systems that estimate the distribution of runtime to order the SLAs [8] or even reject them [46]. Liu et al. [25] solve tenant placement in the cloud given the runtime distribution and SLA penalty. Our market works differently in two aspects. First, our contract consists of multiple target times, which is more flexible than the single deadline implicit in these SLAs. Second, we do not require the consumer to propose an SLA that may be rejected. Instead, the consumer makes a request that is priced by the agent according to their capabilities.

Fine-grained pricing is related to query optimization in distributed databases [22, 14] as we execute subtasks using different virtual machines. However, contract optimization has two objectives (time and cost), while query optimization has only one (time). These two objectives propagate differently in the task graph, making the problem more difficult.

8. DISCUSSION

Our framework can be easily extended to handle applications with different QoS parameters. For example, in long-running services, completion time should be replaced by other parameters like response time in the utility function. These parameters are also uncertain due to unstable cloud performance [26]. While we did not experiment with alternative QoS parameters and different application settings, our market framework is already equipped to handle them with appropriate changes to the utility function.

A meaningful extension to our work is to augment the market to handle varying prices of cloud resources. Our current framework assumes fixed prices for resource configurations. However, fluctuating prices do exist in practice. For example, Amazon EC2 allows agents to bid spot instances with much lower price [2]. Moreover, agents can rent reserved instances directly from Amazon EC2 or through its Reserved Instance Marketplace. These options introduce two additional factors to the market. First, the market needs to account for a supply function $S(\alpha_C, t)$. This means there exist $S(\alpha_C, t)$ instances with lower rate α_C and limited available time t. Second, the framework must consider the starting time of a task, which impacts cost as the rate fluctuates. In this case, agents need to estimate both the supply function and demand function at different points in time. This is not a straightforward extension to our work, and will likely lead to a more complex market model.

9. CONCLUSIONS

In this paper, we propose a new marketplace framework that consumers can use to pay for well-defined database computations in the cloud. In contrast with existing pricing mechanisms, which are largely resource-centric, our framework introduces agent services that leverage a plethora of existing tools for time, cost estimation, and scheduling, to provide consumers with personalized cloud-pricing contracts targeting a specific computational task. Agents price contracts to maximize the utility offered to consumers while also producing a profit for their services. Our market can operate in conjunction with existing cloud markets, as it does not alter the way cloud providers offer and price services. It simplifies cloud use for consumers by allowing them to compare contracts, rather than choose resources directly. The market also allows users to extract more value from public cloud resources, achieving cheaper and faster query processing than naïve configurations, while a portion of this value is earned by the agents as profit for their services. Our experimental evaluation using the AWS cloud platform demonstrated that our market framework offers incentives to consumers, who can execute their tasks more cost-effectively, and to agents, who make profit from providing fair and competitive contracts.

Acknowledgements. This material is based upon work supported by the National Science Foundation under grants IIS-1421322, IIS-1453543, CNS-1129454, and CNS-1012748. We thank Yifei Huang and Bo Jiang for valuable insights and discussions.

10. REFERENCES

[1] A. Aboulnaga, Z. Wang, and Z. Y. Zhang. Packing the most onto your cloud. In *CloudDB*, pages 25–28, 2009.

[2] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing amazon ec2 spot instance pricing. *TEAC*, 1(3):16:1–16:20, 2013.

[3] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala. Interaction-aware scheduling of report-generation workloads. *VLDBJ*, 20(4):589–615, 2011.

[4] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. Zdonik. Learning-based query performance modeling and prediction. In *ICDE*, pages 390–401, 2012.

[5] A. Al Daoud, S. Agarwal, and T. Alpcan. Brief announcement: Cloud computing games: Pricing services of large data centers. In *DISC*, pages 309–310, 2009.

[6] A. Bossenbroek, A. Tirado-Ramos, and P. Sloot. Grid resource allocation by means of option contracts. *ISJ*, 3(1):49–64, 2009.

[7] R. Buyya. Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility. In *CCGRID*, pages 1–1, 2009.

[8] Y. Chi, H. Hacígümüş, W.-P. Hsiung, and J. F. Naughton. Distribution-based query scheduling. *PVLDB*, 6(9):673–684, 2013.

[9] B. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. Parkes, J. Shneidman, A. Snoeren, and A. Vahdat. Mirage: A microeconomic resource allocation system for sensornet testbeds. In *EmNetS*, pages 19–28, 2005.

[10] D. De Roure, C. Goble, and R. Stevens. The design and realisation of the virtual research environment for social sharing of workflows. *FGCS*, 25(5):561–567, 2009.

[11] J. Duggan, O. Papaemmanouil, U. Çetintemel, and E. Upfal. Contender: A resource modeling approach for concurrent query performance prediction. In *EDBT*, pages 109–120, 2014.

[12] A. Floratou, J. M. Patel, W. Lang, and A. Halverson. When free is not really free: What does it cost to run a database workload in the cloud? In *TPCTC*, pages 163–179, 2012.

[13] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. Jordan, and D. Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *ICDE*, pages 592–603, 2009.

[14] S. Ganguly, W. Hasan, and R. Krishnamurthy. Query optimization for parallel execution. In *SIGMOD*, pages 9–18, 1992.

[15] M. Hadji, W. Louati, and D. Zeghlache. Constrained pricing for cloud resource allocation. In *NCA*, pages 359–365, 2011.

[16] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *SOCC*, pages 18:1–18:14, 2011.

[17] S. Ibrahim, B. He, and H. Jin. Towards pay-as-you-consume cloud computing. In *SCC*, pages 370–377, 2011.

[18] C. Joe-Wong and S. Sen. Mathematical frameworks for pricing in the cloud: Revenue, fairness, and resource allocations. *CoRR*, abs/1212.0022, 2012.

[19] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi. Characterizing and profiling scientific workflows. *FGCS*, 29(3):682–692, 2013.

[20] Z. Karampaglis, A. Gounaris, and Y. Manolopoulos. A bi-objective cost model for database queries in a multi-cloud environment. In *MEDES*, pages 19:109–19:116, 2014.

[21] T. Kiefer, H. Schön, D. Habich, and W. Lehner. A query, a minute: Evaluating performance isolation in cloud databases. In *TPCTC*, pages 173–187, 2014.

[22] R. S. G. Lanzelotte, P. Valduriez, and M. Zaït. On the effectiveness of optimization search strategies for parallel execution spaces. In *VLDB*, pages 493–504, 1993.

[23] J. Li, A. C. König, V. Narasayya, and S. Chaudhuri. Robust estimation of resource consumption for sql queries using statistical techniques. *PVLDB*, 5(11):1555–1566, 2012.

[24] Y. Li, E. Lo, M. L. Yiu, and W. Xu. Query optimization over cloud data market. In *EDBT*, pages 229–240, 2015.

[25] Z. Liu, H. Hacigümüş, H. J. Moon, Y. Chi, and W.-P. Hsiung. Pmax: Tenant placement in multitenant databases for profit maximization. In *EDBT*, pages 442–453, 2013.

[26] D. Lorenz and A. Orda. Qos routing in networks with uncertain parameters. *TON*, 6(6):768–778, 1998.

[27] R. Marcus and O. Papaemmanouil. Wisedb: A learning-based workload management advisor for cloud databases. *PVLDB*, 9(10):780–791, 2016.

[28] M. Mihailescu and Y. M. Teo. Dynamic resource pricing on federated clouds. In *CCGrid*, pages 513–517, 2010.

[29] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over the internet-the popcorn project. In *ICDCS*, pages 592–601, 1998.

[30] J. Ortiz, V. T. de Almeida, and M. Balazinska. Changing the face of database cloud services with personalized service level agreements. In *CIDR*, 2015.

[31] J. Ortiz, B. Lee, and M. Balazinska. Perfenforce demonstration: Data analytics with performance guarantees. In *SIGMOD*, pages 2141–2144, 2016.

[32] M. B. Qureshi, M. M. Dehnavi, N. Min-Allah, M. S. Qureshi, H. Hussain, I. Rentifis, N. Tziritas, T. Loukopoulos, S. U. Khan, C.-Z. Xu, and A. Y. Zomaya. Survey on grid resource allocation mechanisms. *JGC*, 12(2):399–441, 2014.

[33] L. Ramakrishnan, J. S. Chase, D. Gannon, D. Nurmi, and R. Wolski. Deadline-sensitive workflow orchestration without explicit resource control. *JPDC*, 71(3):343–353, 2011.

[34] E. Rasmusen. *Games and information : an introduction to game theory*. Blackwell Pub., 2007.

[35] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *PVLDB*, 3(1-2):460–471, 2010.

[36] G. Stuer, K. Vanmechelen, and J. Broeckhove. A commodity market algorithm for pricing substitutable grid resources. *FGCS*, 23(5):688–701, 2007.

[37] Z. Tan and J. R. Gurd. Market-based grid resource allocation using a stable continuous double auction. In *GRID*, pages 283–290, 2007.

[38] M. Tanaka and Y. Murakami. Strategy-proof pricing for cloud service composition. *TCC*, PP(99):1–1, 2014.

[39] K. Vanmechelen and J. Broeckhove. A comparative analysis of single-unit vickrey auctions and commodity markets for realizing grid economies with dynamic pricing. In *GECON*, pages 98–111, 2007.

[40] H. Varian. *Intermediate Microeconomics: A Modern Approach*. W.W. Norton & Company, 2010.

[41] R. Wang. Auctions versus posted-price selling. *Am. Econ. Rev.*, 83(4), 1993.

[42] Y. Wang, A. Meliou, and G. Miklau. A consumer-centric market for database computation in the cloud. *CoRR*, abs/1609.02104, 2016.

[43] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. Analyzing market-based resource allocation strategies for the computational grid. *IJHPCA*, 15(3):258–281, 2001.

[44] K. Wolstencroft, R. Haines, D. Fellows, A. R. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. N. de la Hidalga, M. P. B. Vargas, S. Sufi, and C. A. Goble. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *NAR*, 41(Webserver-Issue):557–561, 2013.

[45] W. Wu, X. Wu, H. Hacigümüş, and J. F. Naughton. Uncertainty aware query execution time prediction. *PVLDB*, 7(14):1857–1868, 2014.

[46] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. HacigümüŞ. Activesla: A profit-oriented admission control framework for database-as-a-service providers. In *SOCC*, pages 15:1–15:14, 2011.

[47] H. Xu and B. Li. Dynamic cloud pricing for revenue maximization. *TCC*, 1(2):158–171, 2013.

[48] Z. Ye, A. Bouguettaya, and X. Zhou. Qos-aware cloud service composition based on economic models. In *ICSOC*, pages 111–126, 2012.

[49] C. S. Yeo, S. Venugopal, X. Chu, and R. Buyya. Autonomic metered pricing for a utility computing service. *FGCS*, 26(8):1368–1380, 2010.

[50] A. Zhou and B. He. Transformation-based monetary cost optimizations for workflows in the cloud. *TCC*, 2(1):85–98, 2014.