Truthful Online Scheduling of Cloud Workloads under Uncertainty

MOSHE BABAIOFF, Microsoft Research, Israel RONNY LEMPEL*, Google, USA BRENDAN LUCIER, Microsoft Research, USA ISHAI MENACHE, Microsoft Research, USA ALEKSANDRS SLIVKINS, Microsoft Research, USA SAM CHIU-WAI WONG*, Microsoft Research, USA

Cloud computing customers often submit repeating jobs and computation pipelines on *approximately* regular schedules, with arrival and running times that exhibit variance. This pattern, typical of training tasks in machine learning, allows customers to partially predict future job requirements. We develop a model of cloud computing platforms that receive statements of work (SoWs) in an online fashion. The SoWs describe future jobs whose arrival times and durations are probabilistic, and whose utility to the submitting agents declines with completion time. The arrival and duration distributions, as well as the utility functions, are considered private customer information and are reported by strategic agents to a scheduler that is optimizing for social welfare.

We design pricing, scheduling, and eviction mechanisms that incentivize truthful reporting of SoWs. An important challenge is maintaining incentives despite the possibility of the platform becoming saturated. We introduce a framework to reduce scheduling under uncertainty to a relaxed scheduling problem without uncertainty. Using this framework, we tackle both adversarial and stochastic submissions of statements of work, and obtain logarithmic and constant competitive mechanisms, respectively.

 $\label{eq:ccs} \mbox{CCS Concepts: \bullet Theory of computation} \rightarrow \mbox{Algorithmic mechanism design; Online algorithms; \bullet Networks \rightarrow Cloud computing.}$

Additional Key Words and Phrases: scheduling, cloud computing, online algorithms, mechanism design

1 INTRODUCTION

Cloud computing platforms provide computational resources of unparalleled scale to their customers. Making the most of this increasing scale involves scheduling the workloads of many customers concurrently using a large supply of cloud resources. Recent years have seen dramatic growth in demand for a particular type of workload: training pipelines for production-grade machine learning models. Such workloads have particular characteristics and challenges that must be addressed by a cloud platform:

• *Uncertain Stochastic Job Requirements*. Machine-learned models deployed to production often entail data processing and training pipelines that run on a regular schedule, *e.g.*, weekly or hourly. The training step might depend on the completion of several data preparation jobs (cleaning data, feature engineering and encoding, etc.) that run in some prerequisite order. This structure makes it possible to predict future jobs and schedule resources in advance. However, the exact timing of any particular job instance will depend on factors such as the size of the training data that may only be revealed at the moment the job is to be executed (and maybe not even then). Thus the cloud

Authors' addresses: Moshe Babaioff, Microsoft Research, Herzliya, Israel, moshe@microsoft.com; Ronny Lempel, Google, Kirkland, WA, USA, rlempel@google.com; Brendan Lucier, Microsoft Research, Cambridge, MA, USA, brlucier@microsoft.com; Ishai Menache, Microsoft Research, Redmond, WA, USA, ishai@microsoft.com; Aleksandrs Slivkins, Microsoft Research, New York City, NY, USA, slivkins@microsoft.com; Sam Chiu-wai Wong, Microsoft Research, Redmond, WA, USA, sam.cw.wong@gmail.com.

^{*}Most work done while the author was at Microsoft.

computing system and the customers may learn the *distribution* of a submitted and upcoming job's duration and the time at which it will become available for execution.

- Latency-dependent Utility. The utility derived by a customer from each instance of a recurring training pipeline will depend on completion time. In some applications, the earlier the refreshed model becomes available for deployment, the better. In other cases, customers may have constant utility up to a strict deadline, and lower or no utility past that. The exact sensitivity to completion time varies greatly across customers as it stems from each customer's business problem and model deployment strategy.
- Information Asymmetry and Incentives. Customers are likely to have more information than the cloud platform about their upcoming jobs, as well as the power to manipulate job requirements. For example, a strategic customer might artificially inflate the size of their training data or introduce unnecessary amounts of concurrency if doing so could result in a better price or lower latency. We therefore consider these attributes to be private information. The platform, which faces strategic agents, must incentivize truthful reporting to ensure that a customer would not gain an advantage by manipulating the predictions of job requirements.¹

How should a cloud platform address these three challenges? Most legacy schedulers are reactive: they have little to no foresight of the arriving workloads, deal with jobs as they arrive, and do not support submission in advance. At the other extreme are schedulers that require workloads to announce their requirements sufficiently in advance, so as to better plan their execution. Neither approach fully addresses the scenario of machine learning training pipelines where jobs are only partially predictable.

1.1 Our Contributions and Techniques

A Model for Stochastic Job Requirements. Our first contribution is a model of cloud scheduling that captures partially predictable job requirements. In our model, jobs are declared to the scheduling system online. Each job comes with concurrency demand and a utility function that determines the value for different completion times, and the scheduler's goal is to maximize total utility (*i.e.*, social welfare). We assume that the total supply of compute nodes is significantly larger than the concurrency demand of any one job. Importantly, a job's specification also includes a *distribution* over possible arrival times (*i.e.*, earliest possible execution time) and duration (*i.e.*, execution time needed to complete). We call this specification a *statement of work* (SoW).²

The platform can make scheduling decisions and declare prices given advance knowledge afforded by the SoWs. However, the arrival time of a job is only revealed online at the moment the job arrives, and the true duration of a job may be only partially known until the moment the job is completed. Jobs are non-preemptable but can be evicted. Payments can depend on realized usage.

The ability to specify a distribution over job requirements instead of reserving resources in advance can significantly impact customer utility. To give a toy example, suppose that a job submitted at time 0 will arrive at some (integral) time $k \le X$, where the probability of arriving at time k is 2^{k-X} for each k < X (and otherwise it arrives at time X). The job always requires a single unit of computation for a single unit of time, but needn't be scheduled immediately upon arrival: it provides utility 2^{X-k} if it completes in round k. If this job is scheduled as soon as it arrives, then it uses

 $^{^{1}}$ While cloud computing customers tend to submit many workloads, we take the common convention that the customers are myopic, optimizing for each job separately.

²Our theoretical model suggests an interface where customers declare distributions directly to the platform. This is an abstraction that highlights customer incentives, since any aspect of the probabilistic information could be manipulated. More generally, a *prediction engine* implemented by the platform could supply some or all of the distributional information on the customer's behalf. We discuss this further in Section 6.

only a single unit of computation and the customer's expected utility is $\Theta(X)$. But if the customer were required to reserve blocks of computation time at the moment of submission, it would be necessary to reserve at least k units to obtain expected utility $\Theta(k)$. With many such jobs, forcing agents to submit deterministic requests would substantially reduce welfare (even though every job is very short). So it would be significantly advantageous to allow agents to submit probabilistic requests and let the platform allocate only the needed resources at the time they are needed (and charge only for the resources used).

Posted-Price Mechanisms with Eviction. We develop a framework for designing truthful online scheduling algorithms for Sows. Our algorithms take the form of posted-price mechanisms that expose a menu of prices, one for each potential allocation of resources. The scheduling algorithm can increase these prices over time as new Sows arrive to the system. When a Sow is revealed for an upcoming job, the scheduler will immediately assign an *execution plan* that maps possible arrival times to job start times, chosen to maximize expected customer utility at the current prices (and subject to eviction probabilities, as described below). Such a scheme incentivizes truthful reporting, since the system optimizes on behalf of the strategic agents. This approach has the important benefit that the system can commit, at the moment a Sow is submitted, to a mapping from realizations to outcome and price.

Posted-price mechanisms for online allocation are not new and have been used in various contexts. A challenge specific to our setting is that, because job requirements are stochastic, a competitive assignment of execution plans will sometimes inadvertently over-allocate the available supply ex post. A common solution is to leave slack when allocating resources to reduce the chance of over-allocation. Unfortunately, this does not suffice to address our problem: since our scheduler is intended to run for an arbitrarily long time horizon, even a low-probability over-allocation event will eventually occur, in which case the platform must evict running jobs and/or cancel future commitments. It is tempting to simply evict all jobs and reset the system in the (very rare) event that an over-allocation occurs. However, this extreme policy could have a significant impact on incentives. A customer who suspects the platform is close to saturation might benefit by misrepresenting their job to finish earlier and thereby avoid an impending eviction.

Instead, our mechanisms evicts jobs in a particular order. Namely, jobs whose SoWs arrived most recently are evicted first. This could include jobs that have not yet started executing, which would be cancelled. This LIFO policy has the important property that the probability a job is evicted is determined at the moment its SoW is submitted, and is independent of future submissions. This allows us to incorporate eviction probabilities into the choice of execution plan, which is crucial for incentives. Indeed, we prove that any algorithm that falls within this framework will incentivize truthful revelation of each SoW, even when the system is close to saturation.

A Reduction to Scheduling Without Uncertainty. We provide a reduction framework for designing mechanisms of the form described above. We consider a relaxed scheduling problem where supply constraints need only hold in expectation over the distribution of job requirements. Hence, there is no danger of saturation, so the problem of designing a competitive online algorithm is significantly simpler. Given an online polytime posted-price algorithm ALG for this relaxed problem, we show how to design a polytime mechanism for the original problem that uses ALG as a guide and (approximately) inherits its performance guarantees.

Theorem 1.1 (Informal). Suppose ALG is a robust posted-price online algorithm that is α -competitive for the relaxed scheduling problem. Then for any $\epsilon>0$, assuming sufficient supply of compute resources, there is a mechanism in our framework that is ϵ -truthful and $\alpha(1+\epsilon)$ -competitive for the original scheduling problem.

As SoWs arrive, our mechanism simulates the progression of ALG (with a relaxed supply constraint) and use ALG's prices when choosing a utility-maximizing execution plan. Our mechanism also tracks the probability of eviction due to saturation and account for eviction when scheduling. When the probability of saturation is sufficiently low, the utility-maximizing choice of allocation approximately coincides between the real and simulated problems. However, when saturation probabilities become too high, the true allocation may diverge from the simulation. To handle this eventuality, we require that algorithm ALG is *robust* in the sense that its welfare degrades gracefully if some job allocations are corrupted by an adversary. In the face of desynchronization, our mechanism will play the role of a corrupting adversary and force ALG to allocate in a manner consistent with the chosen execution plans.

An important technical challenge is that saturation events are correlated across time. Indeed, if the system is currently saturated, it is likely to stay near-saturated in the near future, distorting future allocation decisions. In principle this could lead to a thrashing state where over-allocation begets more over-allocation and the system never recovers. We rule this out, proving that the total realized usage quickly returns to concentrating around its expectation. The proof involves establishing a novel concentration bound for martingales that may be of independent interest.

Online Mechanisms in our Framework. Finally, we provide two example polytime mechanisms that illustrate how to instantiate our framework. First, we consider an *adversarial variant*, where SoWs arrive online in an adversarial but non-adaptive manner.³ In this setting, we design an $O(\log(D_m H))$ -competitive ϵ -truthful online scheduler, where D_m is the maximum duration of any job and the positive job values are normalized to lie in [1, H]. This mirrors known logarithmic-competitive online algorithms for resource allocation, based on exponentially-increasing price thresholds. Indeed we use such an algorithm as an "input" to our reduction.

Second, we consider a *stochastic variant* where the jobs' arrival times are arbitrary but their SoWs are drawn independently from known (but not necessarily identical) distributions. (Since each SoW includes a distribution, the prior information is a distribution over distributions.) We apply our reduction to a variation on a recent O(1)-competitive posted-price mechanism for interval scheduling [15], obtaining an O(1)-competitive ϵ -truthful online scheduler. Unlike the first example, this mechanism will require that a job's duration is revealed at arrival time (i.e., when ready for execution).

Theorem 1.2 (Informal). For any $\epsilon > 0$, assuming a sufficient supply of compute resources, there is an ϵ -truthful, α -competitive online mechanism for scheduling with stochastic job requirements, with $\alpha = O(\log(D_m H))$ for the adversarial variant, and $\alpha = O(1)$ for the stochastic variant. The mechanism for the stochastic variant assumes that job durations are revealed upon job arrival.

1.2 Related Work

Cloud resource management. Cloud resource management has been a very active research topic over more than a decade. Especially relevant is the idea of providing jobs with some form of performance guarantees, often termed Service Level Agreements (SLAs) [17, 27, 36]. To enable the SLAs, the system profiles jobs to estimate their resource requirements, duration, and sometimes even infer their deadlines based on data; see, *e.g.*, [16, 22, 27] and references therein.

Much recent work is dedicated to scheduling machine learning workloads. Particularly relevant are scheduling systems that rely on predicting certain job properties. To highlight a few such systems, Tiresias [23] is a practical system for scheduling ML jobs on GPUs. It uses estimated

³Each SoW still describes a distribution over job requirements and performance is evaluated in expectation over them; it is the SoW specifications that arrive adversarially.

probabilities of resource consumption and job completion times to prioritize resource allocation. Optimus [33] is a job scheduler for deep learning clusters, which builds performance models for estimating the training time as the function of allocated resources. The models are then used to allocate resources dynamically via a centralized optimization problem, to minimize the total completion time. Unlike our model, these schedulers do not account for future jobs and do not address incentives.

Cloud Pricing. The emergence of the cloud business has naturally drawn attention to a variety of economic considerations. Some of the main studied topics include designing proper pricing mechanism (*e.g.*, price structure that leads to efficiency or profit maximization, but still is simple and comprehensible to the end user), how to maximize return on investment (*e.g.*, through spot pricing [1, 32]), how to exploit data for refining the pricing mechanism parameters, etc; see [2, 37] for surveys on cloud pricing.

Let us focus on pricing SLAs between job owners and the cloud provider. [5, 24, 25, 30] posit that jobs have a certain demand for compute resources and a deadline (or more generally, a value function for completion, as in our model), and design incentive-compatible pricing schemes which maximize the social welfare. [8, 26] take a market approach, without zooming in on specific customer. Our paper differs from all these works by explicitly considering the stochastic setting, where both job arrivals and durations are random, a model that is more relevant for ML jobs.

Pricing for machine learning workloads is a relatively new research area. [9] proposes a primal-dual framework for scheduling training jobs, where the resource prices are the dual variables of the framework. Other recent works [12, 31] consider auction-based mechanism for scheduling GPUs, with the general goal of balancing efficiency and fairness. None of these papers models explicitly the stochasticity in job arrival and duration.

Online Scheduling. A rich literature on online scheduling algorithms studies adversarially-chosen jobs that arrive concurrently with execution, and a scheduling algorithm must choose online which jobs to admit. When job values are related to job length, such as when value densities (value over length) are fixed or have bounded ratio, constant-competitive approximations are possible and can be made truthful [28, 35]. When values are arbitrary, there is a lower bound on the power of any randomized scheduler that is polylogarithmic in either (a) the ratio between longest and shortest job lengths, or (b) the ratio between minimum and maximum job values [11], and such bounds have been matched for certain special classes of job values [4]. We likewise obtain a logarithmic approximation, using resource prices that grow exponentially with usage; similar methods were used in incentive compatible online resource allocation going back to [10], and prior to that as an algorithmic method for online routing [29, 34] and load balancing [3, 6].

A technical challenge in the present paper is to limit the impact of cascading failures that arise because overallocation in one round causes an increase of demand for another round. A similar challenge is faced by Chawla et al. [14], who consider a setting where an online scheduler sets a schedule of time-dependent resource prices, and each job is scheduled into the cheapest available timeslot before its deadline. A primary difficulty in this setting is maintaining truthfulness, and further work also explores ways to maintain truthfulness in stateful online resource allocation [13, 18, 20]. Another closely related scheduling mechanism appears in Chawla et al. [15], who consider a Bayesian setting where job requirements are drawn from known distributions and construct a posted-price O(1)-competitive mechanism. Relative to these papers, our model introduces an extra degree of stochasticity where the submitted job requirements are themselves probabilistic.

2 OUR MODEL

We consider an idealized model of a cloud computing platform which captures the challenges discussed above. The platform has C homogeneous computation units called *nodes*. Time proceeds in discrete time-steps (or *rounds*), with t denoting a time-step. At each round, each node can be allocated to some job, for the entire round. There is a finite, known time horizon T. The platform interacts with self-interested job owners, called *agents*. Each agent owns exactly one job; we use index j to denote both.

Each job j requires a fixed number c_j of nodes during its execution, called *concurrency demand*. A job cannot run with fewer nodes nor benefit from additional nodes. The job *arrives* (becomes ready to execute), at *arrival time* a_j . Job j that starts running at time $t \ge a_j$ will be *in execution* and use c_j nodes at each of the d_j consecutive time-steps starting at time t (where d_j is called *duration*). If not interrupted, the job *completes* successfully at time $f_j = t + d_j$. Jobs are *non-preemptable*: they must run continuously in order to finish. The scheduler can *evict* a running job at any time, terminating its execution and reclaiming its nodes.⁵

Each job j brings some value to its owner, depending on whether and when it is completed. This value is $V_j(f_j) \ge 0$ if the job successfully completes (finishes) at time f_j , for some non-increasing function $V_j(\cdot)$ called the *value function*. Otherwise (*i.e.*, if the job is evicted or never starts running) the value is 0. By convention, the completion time is $f_j = \infty$ if the job never completes, and $V(\infty) = 0$.

Each job j is submitted at some time b_j called the *birth* or submission time. At this time, the agent knows the concurrency demand c_j and the value function V_j , but not the arrival time a_j nor the duration d_j . However, the agent knows the joint distribution of (a_j, d_j) , denoted by P_j . No other information is revealed until the job actually arrives, at which time the platform learns a_j . It may also learn some information about d_j when the job arrives, in the form of an observed signal $\sigma_j = \sigma(a_j, d_j)$. No further information about d_j is revealed until the job completes.

Thus, at the job birth time b_j the agent knows the tuple $SoW_j^* = (V_j, c_j, P_j)$, called the true Statement of Work (SoW), and reports a tuple $SoW_j = (V'_j, c'_j, P'_j)$, with the same semantics as the true SoW, called the *reported* SoW. Since our mechanisms incentivize agents to report their true SoWs, it will be the case that $SoW_j = SoW_j^*$.

The number of jobs, their birth times, and their true SoWs constitute the *birth sequence*. The birth sequence is initially unknown to the agents and the platform. We will design schedulers for two settings: adversarial and stochastic. In the *adversarial variant*, the birth sequence is chosen fully adversarially. In the *stochastic variant*, each SoW_j^* is drawn independently from a publicly known (and not necessarily identical) distribution and birth times are chosen adversarially. For a unified presentation, we formally define a problem instance as a distribution over birth sequences.⁷

Once a job j has completed (at time f_j) or been evicted, the agent is charged a payment of $\pi_j \geq 0$. The agent's utility is $u_j = V_j(f_j) - \pi_j$. A job that is never allocated resources has payment (and utility) zero. Agents are risk-neutral and wish to maximize their expected utility. Our mechanisms are ϵ -truthful for some (small) $\epsilon \geq 0$, meaning that for each agent j, given any reported SoWs of the other agents and any realization of the other agents' job requirements (durations and arrival

 $^{^4}$ The finiteness of T is for convenience when defining problem instances. Our guarantees and analysis will not depend on the size of T.

⁵It is possible to reschedule an evicted job, but our mechanisms and benchmarks will not. We therefore treat evictions as permanent for convenience.

⁶For example, if $\sigma_j = d_j$ then the platform learns the duration when the job arrives, and if σ_j is a constant then the platform gains no information about duration. Most of our results hold for arbitrary signals, with the exception of Theorem 5.2.

⁷An adversarial choice of the birth sequence corresponds to an unknown point distribution. In the stochastic variant, the distribution is partially known to the platform.

ALGORITHM 1: Algorithm SchedulerFramework

```
1 Initialize: (\pi^1, \overline{\mathcal{E}}^1) \leftarrow \text{InitInfo}();
2 for each round t do
        If a job j is submitted, choose launch plan L_i as per (2);
3
        for each active job j that arrives at round t do
4
            schedule j to start at time L_i(t, \sigma_i);
 5
        while the current committed load exceeds C do
 6
         Evict/cancel the most-recently-submitted active job;
        Start executing each active job j scheduled to start at t;
8
        for each job j successfully completed at round t do
9
            Charge agent j a payment of \pi^{b_j}(L(a_i), c, d_i);
10
        Update: (\pi^{t+1}, \overline{\mathcal{E}}^{t+1}) \leftarrow \mathsf{UpdateInfo}();
11
```

times), agent j maximizes her expected utility by submitting her true SoW, up to an additive utility loss of at most ϵ . The expectation is over the realization of (a_j, d_j) .

Our mechanism's performance objective is to maximize the total value (or welfare) $\sum_{\text{jobs } j} V_j(f_j)$. We are interested in expected welfare, where the expectation is taken over all applicable randomness.

For comparison, we consider the welfare-maximizing schedule in hindsight, given the value functions, arrival times, and durations of all jobs. The *offline benchmark* is the expected welfare of this schedule on a given problem instance. We are interested in the competitive ratio against this benchmark. Our mechanism is called α -competitive, $\alpha \ge 1$, if its expected welfare is at least $1/\alpha$ of the offline benchmark for each problem instance.

Technical assumptions. We posit some known upper bounds on the jobs' properties: all concurrency demands c_j are at most C_m , all job durations d_j are at most D_m , and all values $V_j(\cdot)$ are at most H. Moreover, $V_j(b_j + S_m) = 0$ for some known S_m ; in words, each agent's value goes down to zero in at most S_m rounds after the job's birth. We assume that $V_j(\cdot) \ge 1$ when positive, i.e., $V_j(\cdot) \in \{0\} \cup [1, H]$.

To simplify notation, we assume that at most one job is submitted at each round. Our algorithm and analysis easily extend to multiple submissions per round, modulo the notation; see Remark 1. The main notations are summarized in Appendix A.

3 THE GENERAL FRAMEWORK

This section presents a general framework for our scheduling mechanism (Algorithm 1), and establishes incentive properties common to all mechanisms in this framework.

Announced info. At each round t, the scheduler updates and announces two pieces of information for jobs that are submitted (and born) in this round: a price menu π^t and estimated failure probabilities $\overline{\mathcal{E}}^t$. Functions π^t and $\overline{\mathcal{E}}^t$ are computed without observing the SoWs for these jobs. The meaning of π^t and $\overline{\mathcal{E}}^t$ is as follows. Suppose job j submitted at time t has reported concurrency demand c_j , realized duration d, and starts executing in some round $t' \geq t$. The scheduler announces the price $\pi^t(t',c,d)$ that would be paid if such job successfully completes, and an estimated probability that it would not complete, denoted $\overline{\mathcal{E}}^t(t',c,d)$. These are announced for all

⁸We use this assumption to achieve a multiplicative competitive ratio. Otherwise, our welfare results would be subject to an extra additive loss. This welfare loss would correspond to that of excluding all jobs with sufficiently small values.

relevant (t', c, d) triples, *i.e.*, for all rounds $t' \in [t, t+S_m]$, demands $c \in [C_m]$, and durations $d \in [D_m]$. By convention, we set $\pi^t(\infty, c, d) = 0$. Prices may vary with t, within two invariants:

- $\pi^t(t', c, d)$ does not decrease with announce time t.
- Costs are non-decreasing in both duration and concurrency: $\pi^t(t', c, d) \leq \pi^t(t', c', d')$ for any $c \leq c'$, $d \leq d'$.

Likewise, estimated failure probabilities may vary over time, but are non-increasing in both duration and concurrency: $\overline{\mathcal{E}}^t(t',c,d) \leq \overline{\mathcal{E}}^t(t',c',d')$ for all $c \leq c', d \leq d'$.

An instantiation of Algorithm 1 should implement InitInfo() and UpdateInfo(). The rest of the algorithm is then fixed.

Launch plans. At each round t, upon receiving the SoW for a given job j, the scheduler computes the *launch plan* L_j for this job, which maps every possible arrival time a_j and signal σ_j (if any) to the start time of the execution. The launch plan may decide to *not* execute the job for some arrival times a_j ; we denote this $L_j(a_j,\sigma_j)=\infty$. The launch plan is binding: job j must start executing at time $L_j(a_j,\sigma_j)$, unless it is *cancelled* beforehand (as explained below).

The choice of a launch plan, described below, is crucial to ensure incentives. For a given launch plan L and a job whose true SoW is (V,c,P), we define the *estimated utility* $\overline{U}_t(L\mid V,c,P)$ as the agent's expected utility under the announced prices π^t , assuming that the estimated failure probabilities are correct. In a formula,

$$\overline{U}_{t}(L \mid V, c, P) = \underset{(a,d) \sim P}{\mathbb{E}} \left[\left(1 - \overline{\mathcal{E}}^{t}(t_{a,\sigma}, c, d) \right) \cdot \left(V(t_{a,\sigma} + d) - \pi^{t}(t_{a,\sigma}, c, d) \right) \right], \tag{1}$$

where $t_{a,\sigma} = L(a, \sigma(a, d))$. We choose a launch plan

$$L_{j} \leftarrow \underset{\text{launch plans } L}{\operatorname{argmax}} \overline{U}_{t}(L \mid V_{j}, c_{j}, P_{j}). \tag{2}$$

so as to maximize the estimated utility given the reported SoW.

Remark 1. For convenience we described Algorithm 1 under the assumption that at most one job is submitted each round. This can be relaxed: if r SoWs are simultaneously submitted at time t, we would choose a launch plan for each job sequentially (in any order), update the announced info after each job, and then move to schedule arriving jobs (line 4) after all r jobs have been handled.

Cancellations and evictions. The scheduler can *cancel* a job that has not yet started executing, or *evict* a job that has. We never restart an evicted or canceled job. A job is called *active* at a given point in time if it has been submitted, but has not yet been completed, cancelled, or evicted. We say that an active job j is *scheduled to start* at round t if it has arrived and $t = L_j(a_j, \sigma_j)$. The *current committed load* is the total concurrency demand, $\sum_j c_j$, of all active jobs j that are executing or scheduled to start in the current round.

If the current committed load is above the total supply C, the scheduler evicts or cancels active jobs in LIFO order of birth time (most recently born first) until the current load is at most C. A job is charged zero payment if it is evicted or cancelled.

Remark 2. The LIFO order is over all active jobs, including jobs that are not scheduled to run in the current round. While this feature is not necessary to address an overbooking failure, it is crucial to our analysis, as explained below.¹⁰

 $[\]overline{{}^{9}$ As a convention, after the last round (t = T) all active jobs are cancelled or evicted.

¹⁰In practice, one might decide *not* to cancel jobs that have not yet started executing. While such a modification would perturb customer incentives, in a real system it might be acceptable as finding beneficial misreporting might be challenging.

We observe that the eviction/cancellation probabilities for a given job are determined at birth/submission time. Formally, let FAIL_j denote the event that job j does not successfully complete, and let \mathcal{H}^t denote the full history of events observed by the algorithm up to (and not including) round t (including all SoWs submitted, launch plans chosen, realized arrivals, job completions, and evictions/cancelations). Also, let $\mathsf{SoW}_{[t,t']}$ denote the collection of SoWs for all jobs submitted in the time interval [t,t'].

LEMMA 3.1. Consider some round t and fix tuple (t, t', c, d). Suppose a job j is submitted in round t with $c_j = c$, and suppose there is a launch plan L such that $\Pr\left[L(a_j, \sigma_j) = t' \text{ and } d_j = d\right] > 0$. Then if launch plan L were chosen for j (i.e., ignoring (2)), then

$$\Pr\left[\mathsf{FAIL}_{j} \mid \mathcal{H}^{t}, \ L(a_{j}, \sigma_{j}) = t', \ d_{j} = d, \ \mathsf{SoW}_{[1,T]}\right]$$
(3)

is determined by \mathcal{H}^t and (t, t', c, d) (and independent of $SoW_{[t,T]}$).

Lemma 3.1 follows immediately from the LIFO ordering: a job j will be evicted/canceled only if the committed load exceeds C even after removing all subsequently-submitted jobs, and this depends only on the launch plans and realizations of previously-submitted jobs. Given Lemma 3.1, we can denote (3) by $\mathcal{E}^t(t',c,d)$, and call it the (true) *failure probability*. We will interpret $\overline{\mathcal{E}}^t(t',c,d)$ as an approximation of $\mathcal{E}^t(t',c,d)$.

While (3) can, in principle, be computed exactly, such computation may be infeasible in practice. We only require the estimates to be approximately correct: we bound the error by some $\mu > 0$, and bound the possible gains from untruthful reporting in terms of μ . Specifically, we assume that, taking the expectation over $\overline{\mathcal{E}}^t$,

$$\mathbb{E}\left[\left|\overline{\mathcal{E}}^{t}(t',c,d) - \mathcal{E}^{t}(t',c,d)\right|\right] < \mu \quad (\forall c,d,t' \ge t). \tag{4}$$

Incentives. Without detailing how the prices are selected and how the estimated success probabilities are computed, we can already guarantee approximate truthfulness. Essentially, this is because launch plans optimize agents' expected utility with respect to the approximate failure probabilities.

Theorem 3.2. Algorithm 1 is $(2\mu H)$ -truthful, where μ bounds the success probability estimation error (as in (4)) and positive job values are normalized to lie in [1, H].

Computation. To compute the optimal L_j in (2) one can separately optimize $L_j(a_j, \sigma_j)$ for each potential arrival time a_j and signal σ_j . This optimization can be done by enumerating over each (a,d) in the support of P_j and each potential start time. One can therefore compute the optimal launch plan in time $O(S_m \cdot |\text{support}(P_j)|)$.

4 REDUCTION APPROACH

We reduce the original problem (henceforth called MainProblem), to its relaxation, RelaxedProblem. The latter is a different but related scheduling problem, where job requirements are fractional rather than uncertain, and the load corresponds to the expected load in MainProblem. Our reduction takes an algorithm for RelaxedProblem, in which over-commitment is never an issue, and use it to solve MainProblem where the system might get saturated in some realizations. In Section 5 we complete this approach by adapting known online resource allocation techniques to solve RelaxedProblem.

4.1 The Relaxed Problem

RelaxedProblem is similar to MainProblem, with these changes:

• Each job j is characterized by a *fractional* SoW, which contains value function $V_j(\cdot) \ge 0$ and concurrency demand c_j as before, but distribution P_j is replaced with k_j tasks τ_1 , ..., τ_{k_j} and weights λ_1 , ..., $\lambda_{k_j} > 0$ with $\sum_i \lambda_i = 1$. Each task τ_i is specified by an arrival time and duration (a_{ij}, d_{ij}) .

- An allocation to job j assigns to each of its tasks τ_i either no resources, or $\lambda_i c_j$ resource units for d_{ij} consecutive timesteps starting no earlier than a_{ij} . Note that $\lambda_i c_j$ might be fractional. Write $x_{ijt} \geq 0$ for the amount of resources allocated to task τ_i at time t, and write f_{ij} for the completion time of this task, or $f_{ij} = \infty$ if it is not completed. The allocation for a single task is called an interval allocation and denoted $x_{ij} = (x_{ijt} : t \in [T])$. The aggregate allocation for job j denoted $x_j = (x_{ijt} : tasks <math>i$, rounds t).
- The value of interval allocation x_{ij} is $\lambda_i V_j(f_{ij})$. The value of the aggregate allocation x_j is $\tilde{V}_j(x_j) = \sum_i \lambda_i V_j(f_{ij})$.
- When a fractional SoW for a given job is submitted, its allocation must be irrevocably decided right away. Tasks cannot be evicted, preempted or cancelled afterwards.
 - The total allocation to all jobs j and tasks i at any time t cannot exceed C, i.e., $\sum_{i,j} x_{ijt} \leq C$.

As before, job's birth times and fractional SoWs comprise a *birth sequence*, which is chosen ahead of time from some distribution over birth sequences. This distribution constitutes a problem instance.

Given an instance I of MainProblem, we construct an instance of RelaxedProblem, denoted Relax(I), in a fairly natural way. For each $SoW_j = (V_j, c_j, P_j)$ in MainProblem, the corresponding fractional SoW has the same V_j and c_j , and tasks $\tau_i = (a_{ij}, d_{ij})$ for each (a_{ij}, d_{ij}) in the support of P_j , with weights $\lambda_i = P_j \left[(a_{ij}, d_{ij}) \right]$. We denote this fractional SoW as Relax(SoW_j).

Any launch plan L_j for job j in MainProblem assigns to each (a_{ij}, d_{ij}) an interval allocation of c_j resources for d_{ij} rounds starting at $L_j(a_{ij}, \sigma(a_{ij}, d_{ij}))$. This corresponds to an interval allocation x_{ij} to each task τ_i in the fractional scheduling problem, in which the resources allocated each round are scaled by λ_i . We will write $x_j(L_j)$ for the aggregate allocation (for all tasks). Note then that $\sum_i x_{ijt}(L_j)$ is the expected usage of resources at time t under launch plan L_j , with respect to probability distribution P_j .

A class of algorithms. Our reduction requires algorithms for RelaxedProblem with the following special structure.

First, an algorithm maintains a price function $\tilde{\pi}$, a.k.a. a *menu*, that assigns a non-negative price to any interval allocation x. The menu can change over time as the algorithm progresses, so we write $\tilde{\pi}^t(x)$ for the price at time t. When a given job j is submitted at time t, the algorithm optimizes the allocation x_j according π^t :

$$x_j \in \underset{\text{aggregate allocations } x_j}{\operatorname{argmax}} \tilde{V}_j(x_j) - \tilde{\pi}^t(x_j),$$
 (5)

where the total job price is $\tilde{\pi}^t(x_j) = \sum_{\text{tasks } i} \tilde{\pi}^t(x_{ij})$. Such algorithms are called *menu-based*.¹¹ We write $\tilde{\pi}^t(t',c,d)$ for the price of allocating c resources for d steps starting at time t'.

Second, the algorithm is measured with respect to the following strong benchmark: for any subset N of jobs, OPT_N is the offline optimal welfare attainable over jobs N in a randomized version of RelaxedProblem, where the choice of allocation to each task can be randomized and the supply constraints need only bind in expectation over this randomization. We write $OPT = OPT_{\{all\ jobs\}}$.

¹¹Note that the argmax in (5) is over *all* aggregate allocations x_j that correspond to launch plans, some of which may violate capacity constraints. To be feasible, the menu must ensure that the output of the argmax stays within the constraints.

ALGORITHM 2: Algorithm 1 via RelaxedProblem

```
Parameters: \epsilon > 0, menu-based, receptive algorithm ALG

// ALG uses supply constraint C' = C \cdot (1 - \epsilon/10)

Require :Oracle for failure probabilities \mathcal{E}^t

// InitInfo() for Line 1 of Algorithm 1

1 Initialize prices \pi^1 same way as ALG; set \overline{\mathcal{E}}^1 \leftarrow 0;

// UpdateInfo() for Line 11 of Algorithm 1

2 if job\ j is submitted in round t then

3 | Report job Relax(SoW<sub>j</sub>) to ALG;

4 | tell ALG: for job j, use allocation x_j(L_j);

// Claim 4.2:\ x_j(L_j) "typically" maximizes (5)

5 Update menu \pi^{t+1} to the updated price menu from ALG;

6 Update \mathcal{E}^{t+1} via the oracle and \overline{\mathcal{E}}^{t+1} \leftarrow f(\mathcal{E}^{t+1}), where

\forall q \geq 0 \ f(q) = q \ \text{if } q > \epsilon/10, \text{ and } 0 \text{ otherwise.}
```

Third, an algorithm should support the following partially adversarial scenario. When a job is born, an adversary can arbitrarily break ties in the choice rule (5). Moreover, the adversary can bypass (5), and instead choose any allocation x_j such that $\tilde{V}_j(x_j) \geq \tilde{\pi}^t(x_j)$. When/if this happens, the algorithm should observe the new x_j and continue. Call such algorithms receptive. The algorithm does not need to compete with OPT. Instead, it only needs to compete with OPT_N, where N is the set of jobs whose allocation satisfies (5) (*i.e.*, is not switched by the adversary). The algorithm is robustly α -competitive, $\alpha \geq 1$ on a given problem instance if for any adversary, the total value generated by the algorithm, $\sum_j \tilde{v}_j(x_j)$ (including jobs scheduled by the adversary) is at least $\frac{1}{\alpha}$ OPT_N.

In summary, an algorithm for RelaxedProblem that is menu-based and receptive uses the following protocol in each round t:

- 1. if a new job j arrives, choose an allocation as per (5),
- 2. replace with the adversarial allocation if applicable,
- 3. update price menu $\tilde{\pi}^t$.

4.2 Reduction to the Relaxed Problem

We instantiate Algorithm 1 using a menu-based, receptive algorithm ALG for RelaxedProblem. This instantiation (Algorithm 2) is competitive for any instance I of MainProblem as long as ALG is robustly-competitive on the relaxed instance Relax(I).

Theorem 4.1. Fix $\epsilon > 0$ such that system's capacity C exceeds $\Omega(C_{\rm m}\,\epsilon^{-2}\log(\epsilon^{-1}+S_{\rm m}))$. Consider an instance I of MainProblem and a menu-based, receptive algorithm ALG for RelaxedProblem. Suppose ALG is robustly α -competitive for the relaxed problem instance Relax(I) and some $\alpha \geq 1$. Then Algorithm 2 with parameter ϵ is $O(\epsilon H)$ -truthful and $\alpha(1 + O(\epsilon))$ -competitive for the original problem instance I. The per-round running time is $O(\epsilon^{-2}\log(\epsilon^{-1}D_{\rm m}S_{\rm m})))$ plus the per-round running time of ALG.

Our reduction proceeds as follows. Formally, in Algorithm 2 we fill out the two unspecified steps in Algorithm 1, InitInfo() and UpdateInfo(). Substantively, we simulate a run of ALG on the relaxed problem instance Relax(I). Whenever a new job j is submitted, we report its fractional

version Relax(SoW_j) to ALG, and use the price menu previously computed by ALG to optimize the launch plan L_j . Then we force ALG to follow the same launch plan for this job: namely, use aggregate allocation $x_j(L_j)$ for job j. If all relevant estimates $\overline{\mathcal{E}}^t$ are zero, this choice just breaks ties in (5):

CLAIM 4.2. In Line 4 of Algorithm 2, suppose $\overline{\mathcal{E}}^t(t',c,d)=0$ for all pairs $(a,d)\in \text{support}(P_j)$ and all times $t'\in[t,t+S_m]$. Then aggregate allocation $x_j(L_j)$ maximizes (5).

We posit oracle access to the (true) failure probabilities \mathcal{E}^t . The simplest version is that the oracle returns exact probabilities. By a slight abuse of notation, we allow the oracle to be ϵ_0 -approximate with probability at least $1-\delta_0$, for some $\epsilon_0=\delta_0=\Theta(\epsilon)$. In Appendix F, we provide an efficient procedure to compute such estimates.

Once we get \mathcal{E}^t from the oracle, we compute the estimates $\overline{\mathcal{E}}^t$ in a somewhat non-intuitive way: we zero out all estimates smaller than a given threshold. Put differently, we ignore failure probabilities if they are sufficiently small. This choice is crucial to "inherit" the performance guarantee of ALG, as we show in the next section.

4.3 Proof of Theorem 4.1

We argue that each job *j* will face low failure probabilities, in the sense of Claim 4.2, with high probability. Then (i) the total value obtained by Algorithm 2 is close to the simulated value obtained in our simulation of ALG, and (ii) the simulated value of ALG is large compared to OPT. We now formalize this intuition.

Fix some birth sequence, and for convenience write V^{ALG} for the total simulated value obtained by ALG in Algorithm 2. Let N be the set of jobs j for which $x_j(L_j)$, from Line 4 of Algorithm 2, maximizes (5). That is, N is the set of jobs whose allocations were not adversarially switched in our simulation of ALG. Then since ALG is robustly α -competitive, we know that $V^{\mathsf{ALG}} \geq \frac{1}{\alpha}\mathsf{OPT}_N$.

Since we actually want to compare V^{ALG} with OPT, we need to show that OPT $_N$ is close to OPT. By Claim 4.2, we will have $j \in N$ whenever all eviction probabilities are sufficiently small for job j. So our goal is to establish that each job is very likely to face very low failure probabilities in Algorithm 2. This is the most technical step in the proof. Intuitively, since ALG constructs allocations subject to a reduced supply constraint C', concentration bounds suggest that it's exponentially unlikely that total realized usage will exceed C in any given round. However, there is correlation between failure probabilities in different rounds. One might therefore worry that even if it takes exponential time for a first eviction to occur, evictions would become more common thereafter. We must therefore bound the impact of correlation across time. This is accomplished by the following lemma (proved in the Appendix).

LEMMA 4.3. Fix any sequence of job birth times and SoWs, and choose any $\lambda > 0$ and $\delta > 0$. If $C' < (1 - \delta)(C - C_m)$, then

$$\Pr\left[\mathcal{E}^{t}(t',c,d) > \lambda\right] < (S_{m})^{2} \cdot \lambda^{-1} \cdot e^{-\Omega\left((C/C_{m}-1)\cdot\delta^{2}/(1+\delta)\right)}$$
(6)

for all $t, t' > t, c \le C_m$, $d \le D_m$, where Pr[] the arrival times and durations for all jobs.

Let γ denote the right-hand side in (6). Assume for now that $\gamma = O(\epsilon)$. If we set $\lambda = \epsilon/10$ (the threshold for $\overline{\mathcal{E}}$ in Algorithm 2), Lemma 4.3 implies that if the oracle for failure probabilities is perfectly accurate, each job will lie in N with probability at least $(1 - \gamma)$. If instead our failure probability oracle is only ϵ_0 -approximate with probability at least $1-\delta_0$, where $\epsilon_0 = \delta_0 = O(\epsilon)$, then we would instead set $\lambda = \epsilon/10 + \epsilon_0$ to conclude that each job will lie in N with probability at least $(1-\gamma-\delta_0)$. This lets us conclude that $OPT_N \geq (1-\gamma-\delta_0)OPT$, and hence $V^{ALG} \geq (1-O(\epsilon))\frac{1}{\alpha}OPT$.

ALGORITHM 3: Adversarial RelaxedProblem

```
require: Capacity constraint C \ge 1

1 For all rounds t, initialize p_t \leftarrow 1/(2D_m), y_t \leftarrow 0;

2 if some job j arrives at time t then

3 Price menu: \tilde{\pi}^t(x_{ij}) = \sum_{t'} p_{t'} \cdot x_{ijt'};

4 Choose some allocation x_j \in \operatorname{argmax}_{x_j} \tilde{V}_j(x) - \tilde{\pi}^t(x_j);

5 Input adversarially chosen allocation x_j (if applicable);

6 for each t' \ge t do

7 y_{t'} \leftarrow y_{t'} + x_{jt'};

8 p_{t'} \leftarrow (4HD_m)^{y_{t'}/C} \cdot (1/(2D_m));
```

The next step is to compare the total value obtained by Algorithm 2 to the simulated value V^{ALG} . The difference between these quantities is that jobs may be evicted in MainProblem, in which case they contribute to the simulated value but not the true realized value. But jobs in N are evicted with probability at most $\epsilon/10 + \epsilon_0 + \delta_0$, by definition of N and the estimation guarantees of our oracle, and by Lemma 4.3 each job lies outside N with probability at most γ . So each job is evicted with probability at most $(\gamma + \epsilon/10 + \epsilon_0 + \delta_0) = O(\epsilon)$. The total value obtained by Algorithm 2 is therefore at least $(1 - O(\epsilon))V^{\text{ALG}} \ge (1 - O(\epsilon))\frac{1}{\alpha}\text{OPT}$.

Finally, we bound the effect of reducing the supply to $C' = C \cdot (1 - \epsilon/10)$ in our simulation. Since OPT is a relaxed benchmark where supply constraints only bind in expectation, this reduction in supply can reduce the value of OPT by a factor of at most $(1 - \epsilon/10)$.

Thus, the total welfare obtained by Algorithm 2 is at least $(1 - O(\epsilon)) \frac{1}{\alpha}$ OPT, as long as $\gamma = O(\epsilon)$. The latter will be true as long as $C > \Omega(C_{\mathfrak{m}} \epsilon^{-2} \log(\epsilon^{-1} + S_{\mathfrak{m}}))$, from the definition of γ . We conclude that Algorithm 2 is $\alpha(1 + O(\epsilon))$ -competitive as required.

5 ROBUST MENU-BASED SCHEDULERS

To complete our solution for MainProblem, we design menu-based, receptive, robustly- α -competitive algorithms for RelaxedProblem, to be used in conjunction with Theorem 4.1. We achieve $\alpha = O(\log(D_{\rm m}H))$ for the adversarial problem variant (when the entire birth sequence is fixed by an adversary), and absolute-constant α for the stochastic problem variant. To both results, per-round running time is ${\rm Poly}(\epsilon^{-1}, S_{\rm m}, |{\rm support}(P_j)|)$. We defer full proofs to the appendix.

Adversarial Variant. We present Algorithm 3. At each round t, it maintains a price per unit of resource at each future round $t' \ge t$. The price function $\tilde{\pi}^t(x_j)$ is a combination of these per-unit prices: $\tilde{\pi}^t(x_j) = \sum_{t'} p_{t'} x_{jt'}$. We then choose a fractional allocation to maximize the expected utility from job j. Subsequently, each price $p_{t'}$ is then updated as a function of $y_{t'}$, the total (fractional) allocation of resources at time t' (including the job just scheduled). Write $p_{t'} = p(y_{t'})$, where

$$p(y_{t'}) = (4HD_{\rm m})^{y_{t'}/C} \cdot 1/(2D_{\rm m}). \tag{7}$$

Note that $p(0) = 1/(2D_m)$, p(C) = 2H, and the prices increase exponentially in usage. These values are tuned so that resources are affordable for any job when usage is 0, but always greater than any customer's willingness to pay when the supply is exhausted.

Theorem 5.1 (adversarial variant). If $C > \Omega(C_{\mathbb{m}} \log(HD_{\mathbb{m}}))$, Algorithm 3 is robustly α -competitive for Relaxed Problem, $\alpha = O(\log(D_{\mathbb{m}}H))$. Plugging it into Algorithm 2 with parameter ϵ such that $C > \Omega(C_{\mathbb{m}} \epsilon^{-2} \log(\epsilon^{-1} + S_{\mathbb{m}}))$, we obtain an $O(\epsilon H)$ -truthful, $O(\alpha)$ -competitive algorithm for Main Problem.

¹²The two problem variants carry over to RelaxedProblem in an obvious way.

PROOF SKETCH. We show that, under our prescribed schedule of price increases, the total value obtained by the algorithm is not much less than the total price of all resources (due to the exponential pricing function), which itself cannot be much less than the total difference in value between the optimum solution and the algorithm's solution (since the optimal-in-hindsight allocation is one of the options considered by Algorithm 3 on Line 3).

Stochastic Variant. We recall the definition of this variant. The number of jobs is fixed, but each job j's SoW is drawn independently from a known distribution F_j of finite support. Once all the SoWs have been drawn, an adversary can choose the submission time for each job, subject to being before the earliest arrival time.

Theorem 5.2 (Stochastic Variant). Suppose $C > \Omega(C_m \log(D_m))$ and $\sigma(a,d) = d$ (i.e., durations are revealed upon job arrival). Then there is a robustly α -competitive algorithm for RelaxedProblem, where α is an absolute constant. Plugging this into Algorithm 2 with parameter ϵ such that $C > \Omega(C_m \epsilon^{-2} \log(\epsilon^{-1} + S_m))$, we obtain an $O(\epsilon H)$ -truthful, $O(\alpha)$ -competitive algorithm for MainProblem.

PROOF SKETCH. We first solve an LP relaxation that encodes the stochastic version of OPT, where supply constraints need only hold in expectation over the distributions F_j . We then need to round this LP solution, online, into a feasible schedule. For this we use a technique from Chawla et al.[15] to partition the LP solution (which is a weighted collection of potential allocations) into disjoint sub-solutions, each of which is associated with a small quantity of resources and can be rounded independently. We then associate each sub-solution with a per-unit price, calculated using the LP solution value, that will be assigned to its corresponding allocations. Using techniques from Prophet Inequalities [15, 19, 21], we show that the posted-price algorithm that allocates in a utility-maximizing way using these prices gives an O(1)-approximation to the LP value.

6 CONCLUSIONS AND FUTURE WORK

This work presented truthful scheduling mechanisms for cloud workloads submitted with uncertainty in jobs' future arrival time and execution duration. These dimensions of uncertainty model the characteristics of repeated jobs and computation pipelines that are prevalent in production workloads. We show how to approach both adversarial and stochastic variants of this model in a unified framework. We reduce to a relaxed problem without uncertainty by employing a particular LIFO eviction policy that minimizes the disruption (to both welfare and incentives) when the available resources are over-allocated in hindsight.

Taken literally, our model suggests an interface where customers provide probabilistic information directly to the platform. This is an abstraction; a more practical implementation would involve a *prediction engine* implemented internally to the platform that predicts the arrival time and duration distributions of regularly submitted jobs. We could then view a SoW as a combination of user-specified input and the predictions, and we would like to ensure that customers are not incentivized to mislead or otherwise confuse the prediction engine. Making this perspective rigorous runs into a subtle three-way distinction between agents' beliefs, the engine's predictions, and the true distributions; we leave this to future work.

Another natural direction for future work is to extend the analysis to richer workload models. For example, *elastic* distributed workloads that may be executed at various concurrency settings, executing faster when utilizing more nodes and slower when running on fewer. Another extension is to preemptable jobs, whose execution may be paused and later resumed without causing the job to fail. Finally, while we focused on obtaining worst-case competitive ratios in this paper, we note that the welfare guarantees in our reduction (Theorem 4.1) actually apply per-instance. It would

be interesting to explore whether this translates into improved performance in well-motivated classes of problem instances.

REFERENCES

- [1] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. Deconstructing amazon ec2 spot instance pricing. ACM Transactions on Economics and Computation (TEAC), 1(3):1–20, 2013.
- [2] May Al-Roomi, Shaikha Al-Ebrahim, Sabika Buqrais, and Imtiaz Ahmad. Cloud computing pricing models: a survey. *International Journal of Grid and Distributed Computing*, 6(5):93–106, 2013.
- [3] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 623–631, 1993.
- [4] Baruch Awerbuch, Yossi Azar, and Serge Plotkin. Throughput-competitive on-line routing. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 32–40. IEEE, 1993.
- [5] Yossi Azar, Inna Kalp-Shaltiel, Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Truthful online scheduling with commitments. In Proceedings of the Sixteenth ACM Conference on Economics and Computation, pages 715–732, 2015.
- [6] Yossi Azar, Bala Kalyanasundaram, Serge Plotkin, Kirk R Pruhs, and Orli Waarts. On-line load balancing of temporary tasks. Journal of Algorithms, 22(1):93–110, 1997.
- [7] Moshe Babaioff, Shaddin Dughmi, Robert Kleinberg, and Aleksandrs Slivkins. Dynamic pricing with limited supply, 2015.
- [8] Moshe Babaioff, Yishay Mansour, Noam Nisan, Gali Noti, Carlo Curino, Nar Ganapathy, Ishai Menache, Omer Reingold, Moshe Tennenholtz, and Erez Timnat. Era: A framework for economic resource allocation for the cloud. In Proceedings of the 26th International Conference on World Wide Web Companion, pages 635–642, 2017.
- [9] Yixin Bao, Yanghua Peng, Chuan Wu, and Zongpeng Li. Online job scheduling in distributed machine learning clusters. In IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pages 495–503. IEEE, 2018.
- [10] Yair Bartal, Rica Gonen, and Noam Nisan. Incentive compatible multi unit combinatorial auctions. In *Proceedings of the 9th conference on Theoretical aspects of rationality and knowledge*, pages 72–87, 2003.
- [11] Ran Canetti and Sandy Irani. Bounding the power of preemption in randomized scheduling. SIAM Journal on Computing, 27(4):993–1015, 1998.
- [12] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16, 2020.
- [13] Shuchi Chawla, Nikhil Devanur, Janardhan Kulkarni, and Rad Niazadeh. Truth and regret in online scheduling. In Proceedings of the 2017 ACM Conference on Economics and Computation, pages 423–440, 2017.
- [14] Shuchi Chawla, Nikhil R Devanur, Alexander E Holroyd, Anna R Karlin, James B Martin, and Balasubramanian Sivan. Stability of service under time-of-use pricing. In Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, pages 184–197, 2017.
- [15] Shuchi Chawla, J. Benjamin Miller, and Yifeng Teng. Pricing for online resource allocation: Intervals and paths. In Timothy M. Chan, editor, Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019, pages 1962–1981. SIAM, 2019.
- [16] Andrew Chung, Subru Krishnan, Konstantinos Karanasos, Carlo Curino, and Gregory R Ganger. Unearthing inter-job dependencies for better cluster scheduling. In 14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20), pages 1205–1223, 2020.
- [17] Carlo Curino, Djellel E Difallah, Chris Douglas, Subru Krishnan, Raghu Ramakrishnan, and Sriram Rao. Reservation-based scheduling: If you're late don't blame us! In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14, 2014.
- [18] Nikhil R Devanur, Kamal Jain, Balasubramanian Sivan, and Christopher A Wilkens. Near optimal online algorithms and fast approximation algorithms for resource allocation problems. *Journal of the ACM (JACM)*, 66(1):1–41, 2019.
- [19] Paul Dutting, Michal Feldman, Thomas Kesselheim, and Brendan Lucier. Prophet inequalities made easy: Stochastic optimization by pricing nonstochastic inputs. SIAM Journal on Computing, 49(3):540–582, 2020.
- [20] Yuval Emek, Ron Lavi, Rad Niazadeh, and Yangguang Shi. Stateful posted pricing with vanishing regret via dynamic deterministic markov decision processes. arXiv preprint arXiv:2005.01869, 2020.
- [21] Michal Feldman, Nick Gravin, and Brendan Lucier. Combinatorial auctions via posted prices. In Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms, pages 123–135. SIAM, 2014.
- [22] Andrew D Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. Jockey: guaranteed job latency in data parallel clusters. In Proceedings of the 7th ACM european conference on Computer Systems, pages 99–112, 2012.

[23] Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A {GPU} cluster manager for distributed deep learning. In 16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19), pages 485–500, 2019.

- [24] Navendu Jain, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters. ACM Transactions on Parallel Computing (TOPC), 2(1):1–29, 2015.
- [25] Navendu Jain, Ishai Menache, Joseph Seffi Naor, and Jonathan Yaniv. A truthful mechanism for value-based scheduling in cloud computing. *Theory of Computing Systems*, 54(3):388–406, 2014.
- [26] Virajith Jalaparti, Ivan Bliznets, Srikanth Kandula, Brendan Lucier, and Ishai Menache. Dynamic pricing and traffic engineering for timely inter-datacenter transfers. In Proceedings of the 2016 ACM SIGCOMM Conference, pages 73–86, 2016
- [27] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Inigo Goiri, Subru Krishnan, Janardhan Kulkarni, et al. Morpheus: Towards automated SLOs for enterprise clusters. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), pages 117–134, 2016.
- [28] Gilad Koren and Dennis Shasha. D-OVER; an optimal on-line scheduling algorithm for overloaded real-time systems. PhD thesis, Inria, 1992.
- [29] Tom Leighton, Fillia Makedon, Serge Plotkin, Clifford Stein, Eva Tardos, and Spyros Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50(2):228–243, 1995.
- [30] Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Efficient online scheduling for deadline-sensitive jobs. In Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures, pages 305–314, 2013.
- [31] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. Themis: Fair and efficient {GPU} cluster scheduling. In 17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20), pages 289–304, 2020.
- [32] Ishai Menache, Ohad Shamir, and Navendu Jain. On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud. In 11th International Conference on Autonomic Computing ({ICAC} 14), pages 177–187, 2014.
- [33] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–14, 2018.
- [34] Serge A Plotkin, David B Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995.
- [35] Ryan Porter. Mechanism design for online real-time scheduling. In *Proceedings of the 5th ACM conference on Electronic commerce*, pages 61–70, 2004.
- [36] Alexey Tumanov, Timothy Zhu, Jun Woo Park, Michael A Kozuch, Mor Harchol-Balter, and Gregory R Ganger. Tetrisched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In *Proceedings of the Eleventh European Conference on Computer Systems*, pages 1–16, 2016.
- [37] Caesar Wu, Rajkumar Buyya, and Kotagiri Ramamohanarao. Cloud pricing models: Taxonomy, survey, and interdisciplinary challenges. ACM Computing Surveys (CSUR), 52(6):1–36, 2019.

A TABLE OF NOTATION

system's computational capacity
concurrency demand of job <i>j</i>
birth time of job <i>j</i>
arrival time of job <i>j</i>
duration of job <i>j</i>
signal about job j duration revealed at arrival time
joint probability distribution over (a_j, d_j)
completion/finish time of job <i>j</i>
value derived by job j if it completes at time t
max concurrency demand of any job.
max duration of any job
max time difference between birth and completion
max value of any job

B PROOF OF THEOREM 3.2

Suppose job j is submitted at time t, with SoW report $SoW_j = (V', c', P')$. For now, assume $\overline{\mathcal{E}}^t = \mathcal{E}^t$ for all t.

If $c' < c_j$ then the job will receive no value from executing, so we can assume that $c' \ge c_j$. Since the job only pays for resources that it uses (and then only if the job successfully completes), and since prices are set at time t, its expected utility is $\overline{U}_t(L_j \mid V_j, c', P_j)$, as per (1). Note that this utility is weakly decreasing as c' increases, since higher c' only increases the price $\pi^t(L_j(a, \sigma(a, d)), c', d)$ and (true) failure probability $\mathcal{E}^t(L_j(a, \sigma(a, d)), c', d)$, for all arrival times a and durations a. Since a it must be utility-maximizing to declare a from this point onward assume that a for a it must be utility-maximizing to declare a from this point onward assume that a from the point of a from the p

The job's expected utility for any given launch plan L is therefore $\overline{U}_t(L \mid V_j, c_j, P_j)$. Note that it depends only on the true SoW, but not on the reported distribution P' nor the value function V'. The job's utility is therefore maximized when the agent reports truthfully.

Now suppose that the estimated failure probabilities are potentially incorrect by up to μ in expectation. Then the expected calculation of utility for any launch plan with non-negative utility can differ by up to μ H from the true utility. Thus the chosen plan can have expected utility up to 2μ H less than the optimal plan, where here the expectation also includes any randomness in the eviction probability estimator.

C PROOF OF LEMMA 4.3

We use the following concentration bound, which strengthens the standard Azuma-Hoeffding inequality. It considers weighted sums of random variables, where the variables and their weights can depend on earlier realizations. Importantly, the probability bound depends on the expected sum of the random variables, but not the number of random variables. This is important for Lemma 4.3, where we need to establish an error bound that is uniform with respect to time and the number of jobs processed by the algorithm. Lemma C.1 is a variation of a bound that appears as Theorem 4.10 in [7]. We omit the proof due to space constraints.

LEMMA C.1. Suppose x_1, \ldots, x_n are Bernoulli random variables and that c_1, \ldots, c_n are real numbers satisfying $0 \le c_i \le C_m$ for each i, where c_i and the distribution of x_i can depend on x_1, \ldots, x_{i-1} . Write $X = \sum_i c_i x_i$ and suppose $E[X] \le M$. Then for any $\delta \le 1$,

$$\Pr[X > (1+\delta)M] < e^{-\Omega(\frac{\delta^2}{1+\delta} \cdot \frac{M}{C_m})}.$$

With Lemma C.1 in hand, we are now ready to prove Lemma 4.3.

PROOF OF LEMMA 4.3. Recall that x_{jt} is the total expected allocation assigned to tasks of job j in the simulated fractional scheduling problem. Fix some arbitrary $\tilde{t} \geq t'$. Choose an arbitrary assignment of execution plans that satisfy the condition $\sum_j x_{j\tilde{t}} \leq C'$, where the execution plan assigned to each job j can depend on the realization of arrival times and durations of previously-submitted jobs.

Write $z_{j\bar{t}}$ for the *realized* usage of resources at time \tilde{t} by job j. Then we know that $x_{j\bar{t}} = E[z_{j\bar{t}}]$, where the expectation is over the arrival and duration of job j, and $z_{j\bar{t}}$ is either 0 or c(j). The distribution of $z_{j\bar{t}}$ is determined by the launch plan assigned to job j, which can depend on the realization of $z_{k\bar{t}}$ for jobs k that were submitted prior to job j. Therefore Lemma C.1 applies to the random variables $\{z_{j\bar{t}}\}_j$ (considered in the order in which jobs are submitted), and by taking $M = C' < (1 - \delta)(C - C_m)$ we conclude

$$\Pr\left[\sum_{j} z_{j\tilde{t}} > C - C_{\mathtt{m}}\right] < e^{-\Omega\left(\frac{\delta^{2}}{1+\delta} \cdot \frac{C - C_{\mathtt{m}}}{C_{\mathtt{m}}}\right)}.$$

Write $A(\tilde{t})$ for the event that $\sum_j z_{j\tilde{t}} > C - C_m$. We then have that

$$\Pr[A(\tilde{t})] < e^{-\Omega(\frac{\delta^2}{1+\delta}, \frac{C-C_m}{C_m})} \quad \text{for any fixed } \tilde{t} \ge t'.$$
 (8)

We note that the probability bound (8) is with respect to all randomness in realizations as evaluated at time 0. To bound $\mathcal{E}^t(t',c,d)$, we instead need to bound the probability of $A(\tilde{t})$ as evaluated at time t, conditioned upon the history of all observations (i.e., job arrival and completion events) up to time t. We therefore need to consider the evolution of $\Pr[A(\tilde{t})]$ from time 0 to time t, then take a union bound over the timesteps \tilde{t} that can impact $\mathcal{E}^t(t',c,d)$. To this end, consider the history of all realizations that occur up to time t. Call this history \mathcal{H} , which is a random variable with a finite support. We can therefore write $\Pr[A(\tilde{t})] = \sum_{\mathcal{H}} \Pr[\mathcal{H}] \Pr[A(\tilde{t})|\mathcal{H}]$. Now, in preparation for taking a union bound, write $B(\tilde{t})$ for the event that $\Pr[A(\tilde{t})|\mathcal{H}] > \lambda/(S_m)$. We then have that

$$\Pr[A(\tilde{t})] = \sum_{\mathcal{H}} \Pr[\mathcal{H}] \Pr[A(\tilde{t})|\mathcal{H}] > \Pr[B(\tilde{t})] \cdot (\lambda/S_m),$$

and hence $\Pr[B(\tilde{t})] < \Pr[A(\tilde{t})]/(\lambda/S_m)$. In other words,

$$\Pr[B(\tilde{t})] < \frac{S_{\text{m}}}{\lambda} \cdot e^{-\Omega(\frac{\delta^2}{1+\delta} \cdot \frac{C-C_{\text{m}}}{C_{\text{m}}})}. \tag{9}$$

Now consider a job that is submitted at time t, requires c units of resources each round, and has (realized) duration d. Regardless of what schedule this job is assigned, it can be evicted only if the total realized usage exceeds C - c (and hence exceeds $C - C_m$) in some round between t and the time at which the job was scheduled to complete, which is at most $t + S_m$. So by a union bound over the events $\{A(t), A(t+1), \ldots, A(t+S_m)\}$ given \mathcal{H} , we have that

$$\mathcal{E}^{t}(t',c,d) \leq \sum_{k=0}^{S_{m}} \Pr[A(t+k)|\mathcal{H}].$$

Thus, in order for $\mathcal{E}^t(t',c,d)$ to be larger than λ , we must have $\Pr[A(t+k)|\mathcal{H}] > \lambda/(S_m)$ for at least one choice of $k \in \{0,\ldots,S_m\}$, which is to say that at least one of the events in $\{B(t),B(t+1),\ldots,B(t+S_m)\}$ occurs. Taking a union bound over these events and applying (9) yields the desired bound:

$$\Pr[\mathcal{E}^t(t',c,d) > \lambda] \leq \sum_{k=0}^{S_{\text{m}}} \Pr[B(t+k)] < \frac{(S_{\text{m}})^2}{\lambda} \cdot e^{-\Omega(\frac{\delta^2}{1+\delta} \cdot \frac{C-C_{\text{m}}}{C_{\text{m}}})}.$$

D PROOF OF THEOREM 5.1

We first note that the allocation x_j chosen by Algorithm 3 for job j is always feasible. To see why, note that if $y_t > C - C_m$ then $p(y_t) > H$. But the maximum value attainable by any allocation of any job that consumes z > 0 units of computation on round t is $H \cdot z$, which would be less than the price paid for round t only. We conclude that if $y_t > C - C_m$ then no further allocation of resources at time t will be made, and hence we will always have $y_t \le C$.

To bound the competitive ratio of Algorithm 3, we will use an argument inspired by dual fitting. To this end, we will compare the value from the obtained solution (x_{ij}) to an appropriate function of the prices. Note that when job j arrives and is allocated x_j , then since the job obtains nonnegative utility we have

$$\tilde{V}_j(x_j) \ge \sum_t x_{jt} p(y_t) \ge \frac{1}{2} \int_{y_t}^{y_t + x_{jt}} p(z) dz$$

where the second inequality follows since $p(z + C_m) \le 2p(z)$ for all z < C, as long as $C > C_m \log(4HD_m)$. Write p_t^* and y_t^* for the prices and total usage, respectively, at the conclusion of the algorithm. Then, summing over all j and integrating the formula in (7),

$$\sum_{j} \tilde{V}_{j}(x_{j}) \ge \frac{1}{2} \sum_{t} \int_{0}^{y_{t}^{*}} p(z) dz = \frac{C}{2 \log 4HD_{m}} \sum_{t} p_{t}^{*} - p(0).$$
 (10)

Now recall the definition of a robustly competitive algorithm, and let N denote the subset of jobs j that are not adversarially allocated. Let $\{z_j\}_j$ denote any (possibly randomized) feasible allocation of the jobs in N. For convenience we will write z_{jt} for the expected allocation at round t under z_j . Let $S \subseteq N$ denote the subset of jobs for which $\mathbb{E}[\tilde{V}_j(z_j)] > 3 \sum_t z_{jt} (p_t^* - p(0))$, and let $T = N \setminus S$. For any $j \in S$, since job j could have been allocated any allocation in the support of z_j , and since prices at the birth of job j can only be lower than (p_t^*) , we conclude from the choice of x_j (on Line 4 of Algorithm 3) and linearity of expectation that

$$\tilde{V}_{j}(x_{j}) \geq \tilde{V}_{j}(x_{j}) - \tilde{\pi}^{b_{j}}(x_{j})$$

$$\geq \mathbb{E}[\tilde{V}_{j}(z_{j})] - \sum_{t} z_{jt} p_{t}^{*}$$

$$\geq \mathbb{E}[\tilde{V}_{j}(z_{j})] - \sum_{t} z_{jt} p(0) - \sum_{t} z_{jt} (p_{t}^{*} - p(0))$$

$$\geq \mathbb{E}[\tilde{V}_{j}(z_{j})] - \frac{1}{2} \mathbb{E}[\tilde{V}_{j}(z_{j})] - \frac{1}{3} \mathbb{E}[\tilde{V}_{j}(z_{j})]$$

$$= \frac{1}{6} \mathbb{E}[\tilde{V}_{j}(z_{j})] \qquad (11)$$

where the second-to-last inequality line from the definition of S and the fact that $\mathbb{E}[\tilde{V}_j(z_j)] \ge 2\sum_t z_{jt} p(0)$ since $p(0) = 1/2D_m$ is half the minimum value density of any job.

Next consider jobs in T, and note that we must have $\sum_{j \in T} z_{jt} \le C$ for each round t, by feasibility. Thus

$$\sum_{j \in T} \mathbb{E}[\tilde{V}_{j}(z_{j})] \leq 3 \sum_{j \in B} \sum_{t} z_{jt} (p_{t}^{*} - p(0))$$

$$\leq 3C \sum_{t} p_{t}^{*} - p(0)$$

$$\leq 6 \log\{4HD_{m}\} \sum_{i} \tilde{V}_{i}(x_{i})$$
(12)

where the last inequality is (10). Combining (11) and (12) yields

$$\begin{split} \sum_{j} \mathbb{E}[\tilde{V}_{j}(z_{j})] &= \sum_{j \in A} \mathbb{E}[\tilde{V}_{j}(z_{j})] + \sum_{j \in B} \mathbb{E}[\tilde{V}_{j}(z_{j})] \\ &\leq (6 \log(4HD_{\mathsf{m}}) + 6) \sum_{j} \tilde{V}_{j}(x_{j}). \end{split}$$

Thus Algorithm 3 is robustly $O(\log(HD_m))$ -competitive.

E PROOF OF THEOREM 5.2

We first recall the statement of Theorem 5.2. We suppose $C > \Omega(C_{\mathrm{m}} \log(D_{\mathrm{m}}))$ and $\sigma(a,d) = d$ (i.e., durations are revealed upon job arrival). Then we claim that there is a robustly α -competitive algorithm for RelaxedProblem, where α is an absolute constant. Plugging this into Algorithm 2 with parameter ϵ such that $C > \Omega(C_{\mathrm{m}} \, \epsilon^{-2} \log(\epsilon^{-1} + S_{\mathrm{m}}))$, we obtain an $O(\epsilon H)$ -truthful, $O(\alpha)$ -competitive algorithm for MainProblem.

We begin the proof by expressing the optimal relaxed fractional assignment as an LP. We will say that an outcome for job j, indexed by ω , is a tuple (j, SoW_j, i, x_{ij}) . The interpretation is that j's realized statement of work (from the support of F_j) is SoW_j , and that task i of SoW_j was provided interval allocation x_{ij} . We will write $SoW(\omega)$, $x(\omega)$, $\tau(\omega)$, etc., for the SoW, allocation, and task associated with ω , respectively. We will also write q_{SoW_j} for the probability of statement of work SoW_j under F_j . Our relaxed LP is then as follows, where the variables (z_ω) are interpreted as the fractional assignment of each outcome ω .

$$\begin{aligned} \max \sum_{\omega} \tilde{V}(\omega) z_{\omega} \\ \text{s.t.} \sum_{\omega} x_{t}(\omega) z_{\omega} &\leq C \quad \forall t \\ \sum_{\substack{\omega: \\ \text{SoW}(\omega) = \text{SoW}_{j}, \\ \tau(\omega) = \tau_{i}}} z_{\omega} &\leq q_{\text{SoW}_{j}} \quad \forall \tau_{i} \in \text{SoW}_{j}, \forall \text{SoW}_{j} \in Supp(F_{j}), \forall j \\ z_{\omega} &\in [0, 1] \quad \forall \omega \end{aligned}$$

Here the first constraint imposes the supply restriction, that the total expected resources allocated over all possible outcomes is at most C. The second constraint is that the total probability assigned to outcomes for a given subtask of a given SoW does not exceed the probability that the SoW is realized from F_i . For a given solution z to this LP, we will write Val(z) for its total value.

We are now ready to describe our approach to computing prices for the menu-based algorithm promised by Theorem 5.2. For this we will use the notion of a "fractional unit allocation" from Chawla et al.[15]. We restate it here in our notation. This involves a slight extension of their definition, since we allow interval allocations to have width up to C_m .

DEFINITION 1. An LP solution (z) is a fractional unit allocation if there exists a partition of the multiset of resources (where each resource in round t has multiplicity C) into bundles $\{B_1, B_2, \dots\}$ and a corresponding partition of job outcomes ω with $z_{\omega} > 0$ into sets $\{A_1, A_2, \dots\}$ such that:

- For each k and $\omega \in A_k$, $x(\omega) \subseteq B_k$
- For each k, $\sum_{\omega \in A_k} w(\omega) z_{\omega} \leq C_m$
- For each k and t, if B_k contains any units of resource from round t, then B_k contains at least C_m units from round t.

Roughly speaking, a fractional unit allocation can be decomposed into disjoint "sub-allocations" that are independent of each other, such that the total fractional weight of each sub-allocation is

at most C_m (the maximum demand of a single job). The third condition ensures that it is always feasible to schedule any single outcome from each set of the partition.

We will make use of the following result from [15], which is implicit in the proof of their Theorem 1.2. We again restate in our notation.¹³

Theorem E.1 ([15]). Suppose $C > C_m \log D_m$. Then for any instance of the stochastic fractional allocation problem, there is a fractional unit allocation z that is an O(1) approximation to the optimal allocation value.

In [15] it is shown how to use the fractional unit allocation from Theorem E.1 to design a static, anonymous bundle pricing menu with high welfare guarantee, for the setting of interval jobs with unit width. That proof makes use of the assumption that all jobs require exactly one unit of resource per unit time. This does not hold in our setting, since (a) we allow jobs to have width up to C_m , and (more crucially) (b) in our setting, each task τ_i has its requirements scaled by λ_i , which can be arbitrarily small. However, as we now show, it is still possible to define a pricing function that guarantees high total value in expectation, at the cost of inflating the resource requirements by a constant factor.

Lemma E.2. For any fractional unit allocation z that is feasible under supply constraint C, there exists a robust menu-based algorithm with supply constraint 2C whose expected welfare is at least $\frac{1}{2}Val(z)$.

PROOF. Let A_k and B_k be the bundles from the fractional unit allocation z. For each A_k define $V(A_k) = \sum_{\omega \in A_k} \tilde{V}(\omega) z_{\omega}$, and write $W(A_k) = \sum_{\omega} w(\omega) z_{\omega}$. That is, $V(A_k)$ and $W(A_k)$ are the total fractional value and weight, respectively, of allocations in A_k . Then for each bundle B_k , we will define the price per unit of B_k to be $p_k = \frac{1}{2W(A_k)}V(A_k)$.

We will now define our price function $\tilde{\pi}^t(x_j)$ for interval allocations (which defines our menubased algorithm). For each bundle B_k , write R_k^t for the fractional weight of allocations to B_k up to time t. Initially all of these fractional weights are zero; that is, $R_k^0 = 0$ for all k. For each k, we say that an interval allocation x_j is *feasible for* B_k at time t, written $x_j \in \mathcal{F}^t(k)$, if $x_j \subseteq B_k$ and $w(x_j) + R_k^t \le 2C_m$. We then define $\tilde{\pi}^t(x_j) = \min_{k:x_j \in \mathcal{F}^t(k)} \{w(x) \cdot p_k\}$. If x_j is not feasible for any B_k then $\pi(x_j) = +\infty$. Note that these menu prices are weakly increasing in job duration and width, and that these prices only ever increase as more jobs are scheduled. They are also well-defined even if some jobs are scheduled arbitrarily (subject to feasibility and non-negative utility), as required by robustness.

Now that our algorithm is defined, we first claim that it generates feasible allocations. Since allocations can only be made to feasible buckets (even adversarially-selected allocations, since non-feasible buckets have infinite price), the schedule will always maintain the property that R_k^t is at most twice C_m for all t. That is, the total width of all allocations to B_k is at most $2C_m$, which means that for all t we have $\sum_{(i,j) \text{ allocated to } B_k} x_{ijt} \leq 2C_m$ which (from the definition of a fractional unit allocation) is at most twice the number of units of time-t resource contained in multiset B_k . Since the sets B_k formed a partition of at most C items per round, we conclude that $\sum_k \sum_{(i,j) \text{ allocated to } B_k} x_{ijt} \leq 2C$, and hence the resulting allocation will be feasible for supply constraint 2C.

We next show that the expected welfare generated by this menu-based allocation is at least $\frac{1}{2}Val(z)$. Recall that the total expected welfare is the sum of the total revenue (payments made)

¹³In [15] it was assumed that $C_{m} = 1$, but the result extends directly to the case of $C_{m} > 1$. Indeed, in the relaxed LP, a task of width greater than 1 can be treated equivalently as a collection of tasks each with width at most 1. And since our bound on total supply is also scaled by C_{m} , the requirement that B_{k} contains at least C_{m} units or none, in each round, corresponds to the fact in [15] that B_{k} contains at least 1 unit or none.

and the total utility of all buyers. For any realization of the jobs' valuations and arrival order, let Z_k denote the event that the total quantity of bundle B_k purchased at the end of the algorithm is at least C_m . The total payment made by all jobs is then

$$Rev = \sum_{k} \Pr[Z_k = 1] p_k \cdot C_m \ge \frac{1}{2} \sum_{k} \Pr[Z_k = 1] V(A_k).$$

Now consider the total utility (value minus payments) obtained by all jobs that are not scheduled adversarially. Each task of each such job will be allocated to a utility-maximizing choice of bundle k for which it is still feasible. Note that if Z_k does not occur, then bucket k will certainly be feasible for any allocation in A_k . For any outcome ω , write k_{ω} for the index k such that $\omega \in A_k$. Then for a job j with realized statement of work SoW_j and task τ_i , the user will obtain expected utility (denoted $u_j(\mathsf{SoW}_j, \tau_i)$) at least

$$\begin{split} &u_{j}(\mathsf{SoW}_{j},\tau_{i}) \\ &\geq E\left[\max_{\substack{\omega: \mathsf{SoW}(\omega) = \mathsf{SoW}_{j}, \\ \tau(\omega) = \tau_{i}}} \mathbb{1}[Z_{k_{\omega}} = 0] \left(\tilde{V}_{j}(x(\omega)) - w(x(\omega))p_{k_{\omega}}\right)^{+}\right] \\ &\geq \frac{1}{q_{\mathsf{SoW}_{j}}} \sum_{\substack{\omega: \mathsf{SoW}(\omega) = \mathsf{SoW}_{j}, \\ \tau(\omega) = \tau_{i}}} \Pr[Z_{k_{\omega}} = 0] z_{\omega} \left(\tilde{V}_{j}(x(\omega)) - w(x(\omega))p_{k_{z}}\right)^{+} \end{split}$$

where the second inequality follows from the feasibility of solution z. Summing over all jobs, SoWs, and tasks, the total utility obtained by all buyers is at least

$$\begin{split} Util &\geq \sum_{j, \mathsf{SoW}_j, \tau_i} q_{\mathsf{SoW}_j} u_j(\mathsf{SoW}_j, \tau_i) \\ &\geq \sum_{j, \mathsf{SoW}_j, \tau_i} \sum_{\omega: \mathsf{SoW}(\omega) = \mathsf{SoW}_j, \\ \tau(\omega) = \tau_i} \Pr[Z_{k_\omega} = 0] z_\omega(\tilde{V}(\omega) - w(\omega) p_{k_\omega})^+ \\ &\geq \sum_k \Pr[Z_k = 0] \sum_{\omega \in A_k} z_\omega(\tilde{V}(\omega) - w(\omega) p_k) \\ &\geq \sum_k \Pr[Z_k = 0] (V(A_k) - W(A_k) p_k) \\ &= \sum_k \Pr[Z_k = 0] (V(A_k) - \frac{1}{2} V(A_k)) \\ &= \frac{1}{2} \sum_k \Pr[Z_k = 0] V(A_k). \end{split}$$

The result now follows by summing the utility and revenue terms.

To complete the proof of Theorem 5.2, we construct a fractional unit allocation z as in Theorem E.1 under constrained supply C' = C/2. We then use this z to construct a robust menu-based pricing method as in Lemma E.2 for total supply 2C' = C. Combining the approximation factors from Theorem E.1 and Lemma E.2, we conclude that the resulting scheduler is O(1)-competitive for the stochastic fractional scheduling problem.

 $^{^{14}}$ It is here where we use the assumption that $\sigma(a,d)=d$. We are allowing the algorithm to allocate each task independently of the other tasks from the same job, which in particular means that tasks with the same arrival time but different runtimes can be scheduled to different start times (and hence runtime is known to the algorithm at submission time).

F ESTIMATING FAILURE PROBABILITIES

In Section 4.2 we described Algorithm 2 assuming access to a relaxed failure probability oracle that is ϵ_0 -approximate with probability at least $1-\delta_0$, where $\epsilon_0=\delta_0=\Theta(\epsilon)$. We now specify the details of this oracle, which we will implement via sampling. When a job j is born at time t, we consider each potential start time $t' \geq t$ and duration d for that job. Recall that the number of such possible pairs (t',d) is bounded. Note that the launch plans of all other jobs are fixed; we only consider variation in the start time of job j. For each possible (t',d), we simulate execution of the resulting schedule T times. In each simulation we realize any residual randomness in the arrival and duration of all jobs that have been born so far, excluding job j, and observe whether job j is evicted given that it starts execution at time t' and runs for d timesteps. Importantly, the failure probability for job j is independent of any jobs that arrive after time t, due to the LIFO eviction order. So, in each simulation, job j fails with probability exactly $\mathcal{E}^t(t',c_j,d)$, independently across simulations. Our estimate, $\tilde{\mathcal{E}}^t(t',c_j,d)$, will be the empirical average over all T simulations. Taking T sufficiently large, the Hoeffding inequality a union bound over all choices of t' and d will imply that our failure probability estimates are sufficiently accurate. The following lemma makes this precise.

LEMMA F.1. Fix $\epsilon_0 > 0$ and $\delta_0 > 0$ and let $T = \log(D_m S_m/\delta_0))\frac{2}{\epsilon_0^2}$. Suppose we take T samples to estimate failure probabilities in the procedure described above, then for each job j the following event occurs with probability at least $1 - \delta_0$: $|\mathcal{E}^t(t', c_j, d) - \tilde{\mathcal{E}}^t(t', c_j, d)| \le \epsilon_0$ for all possible start times t' and durations d for job j.

PROOF. We then have that $\tilde{\mathcal{E}}^t(t',c_j,d)$ is the empirical average of T Bernoulli random variables, each with expectation $\mathcal{E}^t(t',c_j,d)$. Then by the Hoeffding inequality,

$$\Pr[|\tilde{\mathcal{E}}^t(t',c_j,d) - \mathcal{E}^t(t',c_j,d)| > \epsilon_0] < e^{-2T\epsilon_0^2} = \frac{\delta_0}{D_{\text{m}}S_{\text{m}}}.$$

Taking a union bound over all possible choices of (t', d) (of which there are at most $S_m D_m$) concludes the proof.