A Truthful Mechanism for Value-Based Scheduling in Cloud Computing

Navendu Jain[1], Ishai Menache[1], Joseph (Seffi) Naor[2⋆†], and Jonathan Yaniv[2†]

[1] Extreme Computing Group, Microsoft Research, Redmond, WA
[2] Computer Science Department, Technion, Haifa, Israel

Abstract. We introduce a novel pricing and resource allocation approach for batch jobs on cloud systems. In our economic model, users submit jobs with a value function that specifies willingness to pay as a function of job due dates. The cloud provider in response allocates a subset of these jobs, taking into advantage the *flexibility* of allocating resources to jobs in the cloud environment. Focusing on social-welfare as the system objective (especially relevant for private or in-house clouds), we construct a resource allocation algorithm which provides a small approximation factor that approaches 2 as the number of servers increases. An appealing property of our scheme is that jobs are allocated non-preemptively, i.e., jobs run in one shot without interruption. This property has practical significance, as it avoids significant network and storage resources for checkpointing. Based on this algorithm, we then design an *efficient* truthful-in-expectation mechanism, which significantly improves the running complexity of black-box reduction mechanisms that can be applied to the problem, thereby facilitating its implementation in real systems.

1 Introduction

Cloud computing offers easily accessible computing resources of variable size and capabilities. This paradigm allows applications to rent computing resources and services on-demand, benefiting from dynamic allocation and the economy of scale of large data centers. Cloud computing providers, such as Microsoft, Amazon and Google, are offering cloud hosting of user applications under a utility pricing model. The most common purchasing options are pay-as-you-go (or *on-demand*) schemes, in which users pay per-unit resource (e.g., a virtual machine) per-unit time (e.g., per hour). The advantage of this pricing approach is in its simplicity, in the sense that users pay for the resources they get. However, such an approach suffers from two shortcomings. First, the user pays for computation as if it were a tangible commodity, rather than paying for desired performance. To exemplify this point, consider a finance firm which has to process the daily stock exchange data with a deadline of an hour before the next trading day. Such a firm does

⋆ Supported in part by the Google Inter-university center for Electronic Markets and Auctions, and by ISF grants 1366/07 and 954/11.
† Part of this work was done while visiting Microsoft Research.

not care about allocation of servers over time as long as the job is finished by its due date. At the same time, the cloud can deliver higher value to users by knowing user-centric valuation for the limited resources being contended for. This form of value-based scheduling, however, is not supported by pay-as-you-go pricing. Second, current pricing schemes lack a market feedback signal that prevents users from submitting unbounded amounts of work. Thus, users are not incentivized to respond to variation in resource demand and supply.

In this paper, we propose a novel pricing model for cloud environments, which focuses on *quality* rather than *quantity*. Specifically, we incorporate the significance of the completion time of a job, rather than the exact number of servers that the job gets at any given time. In our economic model, customers specify the overall amount of resources (server or virtual machine hours) which they require for their job, and how much they are willing to pay for these resources as a function of due date. For example, a particular customer may submit a job at 9am, specifying that she needs a total of 1000 server hours, and is willing to pay \$100 if she gets them by 5pm and \$200 if she gets them by 2pm. This framework is especially relevant for batch jobs (e.g., financial analytics, image processing, search index updates) that are carried out until completion. Under our scheme, the cloud determines the *scheduling* of resources according to the submitted jobs, the users' willingness to pay and its own capacity constraints. This entire approach raises fundamental issues in mechanism design, as users may try to game the system by reporting false values and potentially increasing their utility. Hence, any algorithmic solution should *incentivize* users to report their true values (or willingness to pay) for the different job due dates.

Pricing in shared computing systems such as cloud computing can have diverse objectives, such as maximizing profits or optimizing system-related metrics (e.g., delay or throughput). We focus in this work on maximizing the social welfare, i.e., the sum of users' values. This objective is especially relevant for private or in-house clouds, such as a government cloud, or enterprize computing clusters.

Our results. We design an efficient truthful-in-expectation mechanism for a new scheduling problem, called the Bounded Flexible Scheduling (BFS) problem, which is directly motivated by the cloud computing paradigm. A cloud containing C servers receives a set of job requests with heterogeneous demand and values per deadline (or due date), where the objective is to maximize the social welfare, i.e., the sum of the values of the scheduled jobs. The scheduling of a job is *flexible*, i.e., it can be allocated a different number of servers per time unit and in a possibly preemptive (non-contiguous) manner, under parallelism thresholds. The parallelism threshold represents the job's limitations on parallelized execution. For every job j, we denote by k_j the maximum number of servers that can be allocated to job j in any given time unit. The maximal parallelism thresholds across jobs, denoted by k, is assumed to be much smaller than the cloud capacity C, as typical in practical settings.

No approximation algorithm is known for the BFS problem. When relaxing the parallelism threshold constraint, our model coincides with the problem of maximizing the profit of preemptively scheduling jobs on a single server. Lawler

[9] gives an optimal solution in pseudo-polynomial time via dynamic programming to this problem, implying also an FPTAS for it. However, his algorithm cannot be extended to the case where jobs have parallelization limits.

Our first result is an LP-based approximation algorithm for BFS that gives an approximation factor of $\alpha \triangleq \left(1 + \frac{C}{C-k}\right)(1+\varepsilon)$ to the optimal social welfare for every $\varepsilon > 0$. With the gap between k and C being large, the approximation factor approaches 2. The running time of the algorithm, apart from solving the linear program, is polynomial in the number of jobs, the number of time slots and $\frac{1}{\varepsilon}$. The design of the algorithm proceeds through several steps. We first consider the natural LP formulation for the BFS problem. Since this LP has a very large integrality gap, we strengthen it by incorporating additional constraints that decrease the this gap. We proceed by defining a reallocation algorithm that converts any solution of the LP to a value-equivalent canonical form, in which the number of servers allocated per job does not decrease over the execution period of the job. Our approximation algorithm then decomposes the optimal solution in canonical form to a relatively small number of feasible BFS solutions, with their average social welfare being an α-approximation (thus, at least one of them is an α-approximation). An appealing property of our scheme is that jobs are allocated non-preemptively, i.e., jobs run in one shot without interruption. This property has practical significance, as it avoids significant network and storage resources for checkpointing the intermediate state of jobs that are distributed across multiple servers running in parallel.

The approximation algorithm that we develop is essential for constructing an efficient truthful-in-expectation mechanism that preserves the α-approximation. To obtain this result, we slightly modify the approximation algorithm to get an exact decomposition of an optimal fractional solution. This decomposition is then used to simulate (in expectation) a "fractional" VCG mechanism, which is truthful. The main advantage of our mechanism is that the allocation rule requires only a *single* execution of the approximation algorithm, whereas known black-box reductions that can be applied invoke the approximation algorithm many times, providing only a polynomial bound on the number of invocations. At the end of the paper, we discuss the process of computing the charged payments.

Related Work. We compare our results to known work in algorithmic mechanism design and scheduling. An extensive amount of work has been carried out in these fields, starting with the seminal paper of Nisan and Ronen [10] (see also [11] for a survey book). Of relevance to our work are papers which introduce *black-box* schemes of turning approximation algorithms to incentive compatible mechanisms, while maintaining the approximation ratio of the algorithm. Specifically, Lavi and Swamy [7] show how to construct a truthful-in-expectation mechanism for packing problems that are solved through LP-based approximation algorithms. Dughmi and Roughgarden [6] prove that packing problems that have an FPTAS solution can be turned into a truthful-in-expectation mechanism which is also an FPTAS. We note that there are several papers that combine scheduling and mechanism design (e.g., [8,1]), mostly focusing on makespan minimization.

Scheduling has been a perpetual field of research in operations research and computer science (see e.g., [5,3,4,12,9] and references therein). Of specific relevance to our work are [4,12], which consider variations of the interval-scheduling problem. These papers utilize a decomposition technique for their solutions, which we extend to a more complex model in which the amount of resources allocated to a job can change over time.

2 Definitions and Notation

In the Bounded Flexible Scheduling **(BFS)** problem, a cloud provider is in charge of a cloud containing a fixed number of C servers. The time axis is divided into T time slots $\mathcal{T} = \{1, 2, \dots T\}$. The cloud provider receives requests from n clients, denoted by $\mathcal{J} = \{1, 2, \dots n\}$, where each client has a job that needs to be executed. We will often refer to a client either as a player or by the job belonging to her. The cloud provider can choose to reject some of the job requests, for instance if allocating other jobs increases its profit. In this model, the cloud can gain profit only by fully completing a job.

Every job j is described by a tuple $\langle D_j, k_j, v_j \rangle$. The first parameter D_j, the *demand* of job j, is the total amount of demand units required to complete the job, where a demand unit corresponds to a single server being assigned to the job for a single time slot. Parallel execution of a job is allowed, that is, the job can be executed on several servers in parallel. In this model we assume that the additional overhead due to parallelism is negligible. However, parallel execution of a job is limited by a threshold k_j, which is the maximal number of servers that can be simultaneously assigned to job j in a single time slot. We assume that $k \triangleq \max_j \{k_j\}$ is substantially smaller than the total capacity C, i.e., $k \ll C$.

Let $v_j : \mathcal{T} \to \mathbb{R}^{+,0}$ be the *valuation function* of job j. That is, $v_j(t)$ is the value gained by the owner of job j if job j is completed at time t. The valuation function v_j is naturally assumed to be monotonically non-increasing in t. The goal is to maximize the sum of values of the jobs that are scheduled by the cloud. In this paper, two types of valuation functions will be of specific interest to us:

• **Deadline Valuation Functions:** Here, players have a deadline d_j which they need to meet. Formally, $v_j(t)$ is a step down function, which is equal to a constant scalar v_j until the deadline d_j and 0 afterwards.

• **General Valuation Functions:** The functions $v_j(t)$ are arbitrary monotonically non-increasing functions.

For simplicity of notation, when discussing the case of general valuation functions, we will set $d_j = T$ for every player. Define $\mathcal{T}_j = \{t \in \mathcal{T} : t \leq d_j\}$ as the set of time slots in which job j can be executed and $\mathcal{J}_t = \{j \in \mathcal{J} : t \leq d_j\}$ as the set of jobs that can be executed at time t.

A *mapping* $y_j : \mathcal{T}_j \to [0, k_j]$ is an assignment of servers to job j per time unit, which does not violate the parallelism threshold k_j[3]. A mapping which fully executes job j is called an *allocation*. Formally, an allocation $a_j : \mathcal{T}_j \to [0, k_j]$ is

[3] For tractability, we assume that the assignment y_j is a continuous decision variable. In practice, non-integer allocations will have to be translated to integer ones, for example by processor sharing within each time interval.

a mapping for job j with $\sum_t a_j(t) = D_j$. Denote by $\mathcal{A}_j$ the set of allocations a_j which fully execute job j and let $\mathcal{A} = \bigcup_{j=1}^{n} \mathcal{A}_j$. Let $s(y_j) = \min\{t : y_j(t) > 0\}$ and $e(y_j) = \max\{t : y_j(t) > 0\}$ denote the start and end times of a mapping y_j, respectively. Specifically, for an allocation a_j, $e(a_j)$ is the time in which job j is completed when the job is allocated according to a_j, and $v_j(e(a_j))$ is the value gained by the owner of job j. We will often use $v_j(a_j)$ instead of $v_j(e(a_j))$ to shorten notations.

3 Approximation Algorithm for BFS

In this section we present an algorithm for BFS that approximates the social welfare, i.e., the sum of values gained by the players. When discussing the approximation algorithm, we assume that players bid truthfully. In Section 4, we describe a payment scheme that gives players an incentive to bid truthfully. We begin this section by describing an LP relaxation for the case of deadline valuation functions and continue by presenting a canonical solution form in which all mappings are Monotonically Non Decreasing (MND) mappings, defined later. This result is then generalized for general valuation functions (Section 3.2). Finally, we give a decomposition algorithm (Section 3.3) which yields an α-approximation to the optimal social welfare of BFS.

3.1 LP Relaxation of BFS with Deadline Valuation Functions

Linear Relaxation Consider the following relaxed linear program. Every variable $y_j(t)$ for $t \in \mathcal{T}_j$ in (LP-D) denotes the number of servers assigned to j at time t. We use y_j to denote the mapping induced by the variables $\{y_j(t)\}_{t \in \mathcal{T}_j}$ and x_j as the completed fraction of job j.

$$
\begin{aligned}
\text{(LP-D)} \quad \max \ & \sum_{j=1}^{n} v_j x_j & & \\
\text{s.t.} \ & \sum_{t \in \mathcal{T}_j} y_j(t) = D_j \cdot x_j & \forall j \in \mathcal{J} & \quad (1)\\
& \sum_{j \in \mathcal{J}_t} y_j(t) \leq C & \forall t \in \mathcal{T} & \quad (2)\\
& 0 \leq y_j(t) \leq k_j x_j & \forall j \in \mathcal{J}, t \in \mathcal{T}_j & \quad (3)\\
& 0 \leq x_j \leq 1 & \forall j \in \mathcal{J} & \quad (4)
\end{aligned}
$$

Constraints (1) and (2) are job demand and capacity constraints. Typically, the parallelized execution constraints would take the form $0 \leq y_j(t) \leq k$. However, the integrality gap in this case can be as high as $\Omega(n)$. Intuitively, (3) "prevents" us from getting bad mappings which do not correspond to feasible allocations. That is, if we would have extended a mapping y_j (disregarding capacity constraints) by dividing every entry in y_j by x_j, we would have exceeded

Reallocate(y)

1. While y contains non-MND mappings
   1.1. Let j be a job generating a maximal(a,b)-violation according to $\preceq$
   1.2. **ReallocationStep**(y,j,a,b)

ReallocationStep(y,j,a,b)

1. Let j' be a job such that $y_{j'}(a) < y_{j'}(b)$
2. $\mathcal{T}_{\max} = \{t \in [a,b] : y_{j'}(t) = y_{j'}(b)\}$
3. $\delta = \max\{y_{j'}(t) : t \in [a,b] \setminus \mathcal{T}_{\max}\}$
4. $\Delta = \min\left\{\frac{y_j(a)-y_j(b)}{1+|\mathcal{T}_{\max}|}, \frac{y_{j'}(b)-y_{j'}(a)}{1+|\mathcal{T}_{\max}|}, y_{j'}(b)-\delta\right\}$
5. Reallocate as follows:
   5.1. $y_{j'}(t) \leftarrow y_{j'}(t) - \Delta$ for every $t \in \mathcal{T}_{\max}$
   5.2. $y_{j'}(a) \leftarrow y_{j'}(a) + \Delta \cdot |T_{\max}|$
   5.3. $y_j(a) \leftarrow y_j(a) - \Delta \cdot |T_{\max}|$
   5.4. $y_j(t) \leftarrow y_j(t) + \Delta$ for every $t \in \mathcal{T}_{\max}$

the parallelization threshold of job j. Before continuing, we mention that there is a strong connection between the choice of (3) and the *configuration LP* for the BFS problem. In fact, (3) can be viewed as an efficient way of implementing the configuration LP. We leave the details to the full version of this paper.

MND Mappings and the Reallocation Algorithm We now present a canonical solution form of solutions for (LP-D), in which all mappings are monotonically non decreasing (defined next). This canonical form will allow us to construct an approximation algorithm for BFS with a good approximation factor.

Definition 1. *A **monotonically non-decreasing (MND)** mapping (allocation) $y_j : \mathcal{T}_j \to [0,k_j]$ is a mapping (allocation) which is monotonically non-decreasing in the interval $[s(y_j), e(y_j)]$.*

MND mappings propose implementation advantages, such as the allocation algorithm being non-preemptive, as well as theoretical advantages which will allow us to construct a good approximation algorithm for BFS. We first present the main result of this subsection:

Theorem 1. *There is a poly(n,T) time algorithm that transforms any feasible solution y of (LP-D) to an equivalent solution that obtains the same social welfare as y, in which all mappings are MND mappings.*

This theorem is a result of the following *reallocation algorithm*. Let y be a feasible solution to (LP-D). To simplify arguments, we add an additional "idle" job which is allocated whenever there are free servers. This allows us to assume without loss of generality that in every time slot, all C servers are in use. We present a reallocation algorithm that transforms the mappings in y to MND mappings. The reallocation algorithm will swap between assignments of jobs to servers, without changing the completed fraction of every job (x_j), such that no completion time of a job will be delayed. Since the valuation functions are

deadline valuation functions, the social welfare of the resulting solution will be equal to the social welfare matching y. Specifically, an optimal solution to (LP-D) will remain optimal. We introduce some definitions and notations prior to the description of the reallocation algorithm.

Definition 2. *Job j generates an (a,b)-**violation**, $a<b$, if $y_j(a) > y_j(b) > 0$. Violations are weakly ordered according to a binary relation $\preceq$ over $\mathcal{T}\times\mathcal{T}$:*

$$(a,b) \preceq (a',b') \;\Leftrightarrow\; b<b' \text{ or } (b=b') \wedge (a \leq a') \tag{5}$$

Note that there can be several maximal pairs (a,b) according to $\preceq$.

Given a solution y to (LP-D), our goal is to eliminate all (a,b)-violations in y and consequently remain with only MND mappings, keeping y a feasible solution to (LP-D). The reallocation algorithm works as follows: In every step we try to eliminate one of the maximal (a,b)-violations, according to the order induced by $\preceq$. Let j be the job generating this maximal (a,b)-violation. The main observation is that there must be some job j' with $y_{j'}(a) < y_{j'}(b)$, since in every time slot all C servers are in use. We apply a *reallocation step*, which tries to eliminate this violation by shifting workload of job j from a to later time slots (b in particular), and by doing the opposite to j'. To be precise, we increase y_j in time slots in $\mathcal{T}_{\max}$ (line 2) by a value $\Delta > 0$ (line 4), and increase $y_{j'}(a)$ by the amount we decreased from other variables. We note that if we do not decrease $y_{j'}$ for all time slots in $\mathcal{T}_{\max}$, we will generate $(\tilde{a},b)$-violations for $a<\tilde{a}$ and therefore the reallocation algorithm may not terminate.

We choose Δ such that after calling the reallocation step either: 1. $y_j(a) = y_j(b)$ 2. $y_{j'}(a) = y_{j'}(b)$ 3. The size of $\mathcal{T}_{\max}$ increases. In the second case, if the (a,b)-violation hasn't been resolved, there must be a different job j'' with $y_{j''}(a) < y_{j''}(b)$, and therefore we can call the reallocation step again. In the third case, we simply expand $\mathcal{T}_{\max}$ and recalculate Δ. The reallocation algorithm repeatedly applies the reallocation step, choosing the maximal (a,b)-violation under $\preceq$, until all mappings become MND mappings. The following lemma guarantees the correctness of the reallocation algorithm.

Lemma 1. *Let y be a feasible solution of (LP-D) and let j be a job generating a maximal (a,b)-violation over $\preceq$. Denote by $\tilde{y}$ the vector y after calling ReallocationStep(y,j,a,b) and let $\left(\tilde{a},\tilde{b}\right)$ be the maximal violation in $\tilde{y}$. Then:*

1. *$\tilde{y}$ is a feasible solution of (LP-D).*
2. *$\left(\tilde{a},\tilde{b}\right) \preceq (a,b)$*
3. *No new (a,b)-violations are added to $\tilde{y}$.*

The reallocation algorithm runs in $poly(n,T)$ time. To show this, consider a potential function which is the total number of violations. The reallocation algorithm resolves at least one violation after at most nT calls to the reallocation step. The maximal initial number of such violations is bounded by $O\left(nT^2\right)$ and a reallocation step can be efficiently implemented, proving the statement.

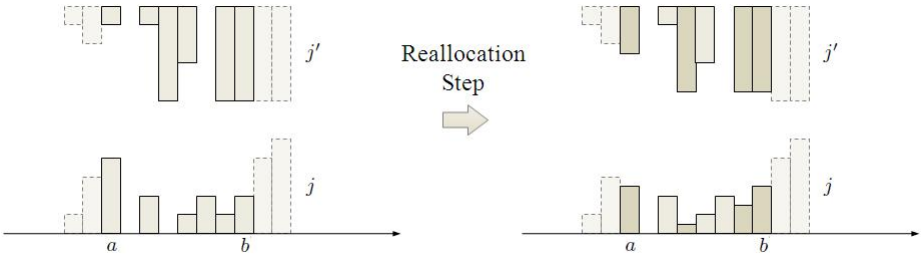

Fig. 1. Resolving an (a, b)-violation generated by j, with $\mathcal{T}_{\max} = \{b, b-1, b-4\}$. X-axis represents time.

3.2 Extension to General Valuation Functions

To extend the results presented so far to the case of general valuation functions, we expand (LP-D) by splitting every player into T subplayers, one for each end time, each associated with a deadline valuation function. Formally, every player j will be substituted by T subplayers $j^1, j^2, \ldots j^T$, all with the same demand and parallelization bound as j. For ease of notation, we denote by $y_j^e(t)$ the variables in the linear program matching subplayer j^e, and use similar superscript notations henceforth. For every subplayer j^e, we set $v_j^e = v_j(e)$ and $d_j^e = e$. Apart from demand and capacity constraints, we include constraint (3) for every subplayer j^e and add an additional set of constraints:

$$\sum_{e \in \mathcal{T}} x_j^e \leq 1 \qquad \forall j \in \mathcal{J} \tag{6}$$

This is indeed a relaxation of BFS (we can map an allocation a_j in a BFS solution to the subplayer $j^{e(a_j)}$). The reallocation algorithm does not change the values x_j^e, thus it will not violate (6). We note that these results can be extended to the case where valuation functions are non-monotone. From this point on, we refer to **(LP)** as the relaxed linear program for general valuation functions, after adding (6). When applying results to deadline valuation functions settings, every player j will be viewed as a single subplayer j^{d_j} (making (6) redundant).

3.3 Decomposing an Optimal MND Fractional Solution

The approximation algorithm presented in this section constructs a set of feasible solutions to BFS from a fractional optimal solution to (LP) given in the canonical MND form. The algorithm is similar to the coloring algorithm used in [4,12] for the weighted job interval scheduling problem. The first step of the algorithm constructs a multiset $\mathcal{S} \subset \bigcup_{j=1}^{n} \mathcal{A}_j$ of allocations out of an optimal solution of (LP) given in MND form and then divides the allocations in $\mathcal{S}$ into a set of feasible solutions to BFS.

Coloring Algorithm($\mathcal{S}$)

1. Sort the MND allocations $a \in \mathcal{S}$ according to $e(a)$ in descending order.
2. For every MND allocation a in this order
   2.1 Color a in some color c such that c remains a feasible integral solution.

Step I: Construction of $\mathcal{S}$. Let N be a large number to be determined later. Consider a job j which is substituted by a set of subplayers $j^1, j^2, \ldots, j^T$ (or a single subplayer j^{d_j} for the case of deadline valuation functions). Let y be an optimal solution of (LP) after applying the reallocation algorithm. For every subplayer j^e, let a_j^e be the allocation corresponding to y_j^e, defined: $a_j^e(t) = \frac{y_j^e(t)}{x_j^e}$ for every $t \in \mathcal{T}_j$. Note that a_j^e is an allocation by the definition of x_j^e and by (3). We construct $\mathcal{S}$ as follows: Let $\bar{x}_j^e$ denote the value x_j^e rounded up to the nearest integer multiplication of $\frac{1}{N}$. For every subplayer j^e, add $N\bar{x}_j^e$ copies of a_j^e to $\mathcal{S}$.

Step II: Coloring Allocations. The coloring algorithm will color copies of MND allocations in $\mathcal{S}$ such that any set of allocations with identical color will induce a feasible integral solution to BFS. Let $1, 2, ..., COL$ denote the set of colors used by the coloring algorithm, described above. We use $a \in c$ to represent that an allocation a is colored in color c. Given a color c, let $c(t) = \sum_{a \in c} a(t)$ denote the total load of MND allocations colored in c at time t. The following two lemmas prove that the number of colors used is relatively small. This allows us to construct an α-approximation algorithm in Theorem 2.

Lemma 2. *Consider an iteration after some allocation $a \in \mathcal{S}$ is colored. Then, for every color c, $c(t)$ is monotonically non-decreasing in the range $[1, e(a)]$.*

Proof (Sketch). Since the sum of MND vectors is MND. Proof by induction. □

Lemma 3. *If $COL = N \cdot \left(1 + \frac{C}{C-k}\right)\left(1 + \frac{nT}{N}\right)$, then when the algorithm handles an allocation $a \in \mathcal{S}$ there is always a free color c in which a can be colored.*

Proof (Sketch). Consider the point when a copy of an allocation a_j is colored. Job j is associated with at most $N\left(1 + \frac{nT}{N}\right) - 1$ copies other than a_j (since we rounded up the values x_j^e. For any color c that cannot be used due to capacity constraints we must have: $c(e(a_j)) \geq C - k$. Thus, there is always a free color in which a_j can be colored. □

Theorem 2. *There is a poly $\left(n, T, \frac{1}{\varepsilon}\right)$ time approximation algorithm that given an optimal solution to (LP) returns an $\alpha \triangleq \left(1 + \frac{C}{C-k}\right)(1 + \varepsilon)$ approximation to BFS for every $\varepsilon > 0$.*

Proof (Sketch). Set $N = \frac{nT}{\varepsilon}$. The social welfare obtained by the best color out of the COL colors is at least: $\frac{N \cdot \sum_{j^e} v_j \bar{x}_j^e}{COL} \geq \frac{N \cdot OPT^*}{COL} = \frac{OPT^*}{\alpha}$ □

4 Truthfulness-in-Expectation

Up until now we have assumed that players report their true valuation functions to the cloud provider and that prices are charged accordingly. However, in reality, players may choose to untruthfully report a valuation function b_j which differs from their true valuation function v_j if they may gain from it. In this section, we construct an efficient mechanism that charges costs from players such that reporting their valuation function untruthfully cannot benefit them. Unlike known black-box reductions for constructing such mechanisms, our construction calls the approximation algorithm only *once*, significantly improving the complexity of the mechanism.

We begin by introducing the common terminology used in mechanism design. Every participating player chooses a type out of a known type space. In our model, players choose a valuation function v_j out of the set of monotonically non-increasing valuation functions (or deadline valuation functions) to represent its true type. Denote by V_j the set of types from which player j can choose and let $V = V_1 \times \cdots \times V_n$. For a vector v, denote by v_{-j} the vector v restricted to entries of players other than j and denote V_{-j} accordingly. Let $\mathcal{O}$ denote the set of all possible outcomes of the mechanism and let $v_j(o)$ for $o \in O$ represents the value gained by player j under outcome o. A *mechanism* $\mathcal{M} = (f, p)$ consists of an allocation rule $f : V \to \mathcal{O}$ and a pricing rule $p_j : V \to \mathbb{R}$ for each player j. Players report a bid type $b_j \in V_j$ to the mechanism, which can be different from their true type v_j. The mechanism, given a reported type vector $b = (b_1, \ldots, b_n)$ computes an outcome $o = f(b)$ and charges $p_j(b)$ from each player. Each player strives to maximize its utility: $u_j(b) = v_j(o) - p_j(b)$, where o_j in our model is the allocation according to which job j is allocated, if at all. Mechanisms such as this, where the valuation function does not consist of a single scalar are called *multi-parameter* mechanisms. Our goal is to construct a multi-parameter mechanism where players benefit by declaring their true type. Another desired property is that players do not lose when truthfully reporting their values.

Definition 3. *A deterministic mechanism is* ***truthful*** *if for any player j, reporting its true type maximizes $u_j(b)$. That is, given any bid $b_j \in V_j$ and any $v_{-j} \in V_{-j}$, we have:*

$$u_j((v_j, v_{-j})) \geq u_j((b_j, v_{-j})) \tag{7}$$

where $v_j \in V_j$ is the true type of player j. A randomized mechanism is ***truthful-in-expectation*** *if for any player j, reporting its true type maximizes the expected value of $u_j(b)$. That is, (7) holds in expectation.*

Definition 4. *A mechanism is* ***individually rational (IR)*** *if $u_j(v)$ does not receive negative values when player j bids truthfully, for every j and $v_{-j} \in V_{-j}$.*

4.1 The Fractional VCG Mechanism

We start by giving a truthful, IR *fractional* mechanism that can return a fractional allocation, that is, allocate fractions of jobs according to (LP):

1. Given reported types $b_j : \mathcal{T} \to \mathbb{R}^{+,0}$, Solve (LP) and get an optimal solution y^*. Let $o \in \mathcal{O}$ be the outcome matching y^*.
2. Charge $p_j(b) = h_j(o_{-j}) - \sum_{i \neq j} b_i(o_i)$ from every player j, where h_j is any function independent of o_j.

This is the well known VCG mechanism. Recall that (LP) maximizes the social welfare, i.e., the sum of values gained by all players. Assuming all other players act truthful, player j gains $u_j(b) = OPT^* - h_j(o_{-j})$ by bidding truthfully and therefore the mechanism is optimal, since deviating can only decrease $\sum_i v_i(o)$. Note that by dividing both valuation functions and charged prices by some constant, the fractional VCG mechanism remains truthful. This will be useful later on. Individual rationality of the fractional VCG mechanism is obtained by setting the functions h_j according to Clarke's pivot rule [11].

4.2 A New Efficient Truthful-in-Expectation Mechanism

Lavi and Swamy [7] give a black-box reduction for combinatorial auction packing problems from constructing a truthful-in-expectation mechanism to finding an approximation algorithm that verifies an integrality gap of the "natural" LP for the problem. Their reduction finds an exact decomposition of the optimal fractional solution (scaled down by some constant β) into a distribution over feasible integer solutions. By sampling a solution out of this distribution and charging payments according to the fractional VCG mechanism (scaled down by β), they obtain truthfulness-in-expectation. The downside of the reduction given in [7] is that the approximation algorithm A is used as a separation oracle for an additional linear program used as part of the reduction, making their construction inefficient. We follow along the lines of [7] in order to construct a truthful-in-expectation mechanism for the BFS problem, and show how to achieve the same results as [7] by calling our approximation algorithm once.

Recall that the algorithm from Theorem 2 constructs a set of feasible solutions to BFS out of an optimal solution to LP. Ideally, we would have wanted to replace the exact decomposition found by [7] with the output of our decomposition algorithm (by drawing one of the colors uniformly). However, this does not work since our decomposition is not an exact one, because the values x_j^e have been rounded up to $\bar{x}_j^e$ prior to the construction of $\mathcal{S}$.

To overcome this issue, we use a simple alternative technique to round the entries in x to integer multiplications of $\frac{1}{N}$. We construct a vector $\tilde{x}$ such that $\mathbb{E}\left[\tilde{x}_j^e\right] = x_j^e$ for every subplayer j^e, as follows: Assume that $x_j^e = \frac{q}{N} + r$ for $q \in \mathbb{N}$ and $0 \leq r < \frac{1}{N}$. Then, set $\tilde{x}_j^e = \frac{q+1}{N}$ with probability $N \cdot r$ and $\tilde{x}_j^e = \frac{q}{N}$ otherwise. Note that $\mathbb{E}\left[\tilde{x}_j^e\right] = x_j^e$ as required. Now, we construct $\mathcal{S}$ out of $\tilde{x}$ and call the coloring algorithm. By uniformly drawing one of the colors c and scheduling jobs according to the allocations colored in c, we obtain an expected welfare of: $\mathbb{E}\left[\frac{N}{COL}\sum_{j^e} v_j \tilde{x}_j^e\right] = \frac{OPT^*}{\alpha}$. By charging fractional VCG prices, scaled down by α, we obtain truthfulness-in-expectation. Notice that this mechanism is not individually rational, since unallocated jobs may be charged. Lavi and Swamy

[7] solve this problem by showing how to modify the pricing rule so that the mechanism will be individually rational. Notice that the number of colors used by the coloring algorithm must always be COL, even though it is an upper bound on the number of colors needed. Otherwise, players might benefit from reporting their valuation functions untruthfully by effecting the number of solutions.

Theorem 3. *There is a truthful-in-expectation, individually rational mechanism for BFS that provides an expected α-approximation of the optimal social welfare.*

Finally, we discuss the process of computing the payments $p_j(b)$. Note that to directly calculate the payments charged by VCG, one must solve a linear program for every player j. [2] describes an implicit pricing scheme that requires only a single invocation of the approximation algorithm to construct both an allocation rule and pricing rules of a truthful-in-expectation mechanism. This result can be plugged into our mechanism, thus decreasing the number of calls to our approximation algorithm to one. However, their scheme induces a mechanism that is only individually rational in expectation (specifically, it may charge negative prices) and causes a multiplicative (constant) loss to social welfare.

References

1. A. Archer and Éva Tardos. Truthful mechanisms for one-parameter agents. In *FOCS*, pages 482–491, 2001.
2. M. Babaioff, R. Kleinberg, and A. Slivkins. Truthful mechanisms with implicit payment computation. In *EC*, pages 43–52, 2010.
3. A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *JACM*, 48:1069–1090, 2001.
4. A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM Journal of Computing*, 31(2):331–352, 2001.
5. P. Brucker. *Scheduling Algorithms*. Springer, 4th edition, 2004.
6. S. Dughmi and T. Roughgarden. Black-box randomized reductions in algorithmic mechanism design. In *FOCS*, pages 775–784, 2010.
7. R. Lavi and C. Swamy. Truthful and near-optimal mechanism design via linear programming. In *FOCS*, pages 595–604, 2005.
8. R. Lavi and C. Swamy. Truthful mechanism design for multi-dimensional scheduling via cycle monotonicity. In *EC*, 2007.
9. E. L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Oper. Research*, 26:125–133, 1991.
10. N. Nisan and A. Ronen. Algorithmic mechanism design. In *STOC*, 1999.
11. N. Nisan, T. Roughgarden, Éva Tardos, and V. V. Vazirani. *Algorithmic game theory*. Cambridge University Press, 2007.
12. C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *SODA*, pages 879–888, 2000.