Truthful Mechanisms for Competitive Reward-Based Scheduling

Lena Mashayekhy, *Student Member, IEEE,* Nathan Fisher, *Member, IEEE,* and Daniel Grosu, *Senior Member, IEEE*

Abstract—We consider a competitive environment for reward-based scheduling of periodic tasks, where the execution of each task consists of a mandatory and an optional part. Each task obtains a value if the processor successfully schedules all its mandatory part, and also an additional reward value if the processor successfully schedules a part of its optional execution. Each task is owned by a self-interested agent who has multiple choices for its requests based on its optional part. We model the reward-based scheduling problem by considering such multi-minded agents, and specify a bidding language for representing their requests. However, the agent may try to manipulate the system to obtain an unfair allocation on the processor. We address this challenge by designing novel truthful mechanisms in which it is always in the agent's best interest to report their true task characteristics. We propose two truthful mechanisms (an exact and approximate) for selecting a feasible subset of agents and an allocation of optional execution that maximizes the total reward obtained by the selected tasks. To address the pseudo-polynomial complexity of the exact mechanism, we show that our proposed approximate mechanism is a polynomial-time approximation scheme (PTAS). Our extensive experiments show that our proposed approximation mechanism is capable of finding near-optimal solutions efficiently while guaranteeing truthfulness.

Index Terms—Reward-Based Scheduling; Periodic Task Systems; Competitive Environments; Mechanism Design.

1 Introduction

Over the past two decades, the increased connectivity between real-time embedded devices has spurred a shift in real-time system design and research from traditionally closed and independent systems to more open and more interconnected systems. In these increasingly open systems, a designer might not know at design-time the characteristics of the set of tasks that will execute in the system. Thus, the system may potentially be executing in situations where it is not possible to meet all the execution demands of the users and must choose which execution should complete and which should be discarded. To address such open system scenarios, real-time system researchers have developed several useful and interesting frameworks for allocating processing time in the presence of overload, such as *reward-based scheduling* [1], *IRIS* [2], *Q-RAM* [3], and many others. Even more recent research has applied these frameworks in the design of open systems; for example, Kim [4] applied reward-based scheduling to manage QoS allocation in a cluster and grid computing setting. While these frameworks have successfully enabled designers to determine the optimal allocation of system execution under overload in an open system, all of these frameworks implicitly rely upon the assumption that users are cooperative by truthfully expressing their timing requirements and the value that they place upon meeting these requirements. However, any robust system design must also consider the possibility that users will lie about their requirements in order to obtain a larger allocation of processing resources. Unfortunately, these previously-proposed frameworks are ill-suited to deal with such a possibility.

In this paper, we consider reward-based scheduling for periodic tasks in which there is a reward associated with each task's execution. Each task's execution is composed of a mandatory and an optional part. Each job of a task must complete both its mandatory execution, as well as any allocated optional execution, by the job's deadline, while a non-decreasing reward function is associated with the execution of the optional part. A reward-based scheduling model is appropriate in settings where real-time tasks can obtain more precise results with additional computation. There are several applications that can benefit from extra processing. For example, the amount of processing may affect the video quality of a multimedia application: more processing yields a better quality video and thus, more reward for the viewer. Another example of such applications is realtime cloud services [5], where a user specifies the least amount of quality of service in the service level agreement. Additional processing power may lead to faster execution time of the user's job, and would increase the user quality of service. In this paper, we take a first step towards the design of overloaded systems in an open and competitive environment by proposing a *competitive reward-based scheduling* framework that utilizes techniques from algorithmic game theory to ensure a fair allocation of resources in open real-time systems even in the presence of uncooperative users.

We extend the traditional reward-based scheduling model [1] to the competitive environment setting where

• *L. Mashayekhy, N. Fisher, and D. Grosu are with the Department of Computer Science, Wayne State University, Detroit, MI, 48202.*
E-mail: mlena@wayne.edu, fishern@wayne.edu, dgrosu@wayne.edu

each task may be owned by a separate agent. Each agent knows the characteristics of its own task and reports, to the processor owner, the task's period, mandatory execution requirement, optional execution and the "reward" that agent obtains from execution of the mandatory part plus any optional part. Each agent submits these characteristics of its task to the processor owner in the form of K bids, where K represents the different choices for its requested optional execution and its corresponding reward. Such agents submitting K bids are called K-minded bidders. The processor owner must determine (based on each task's reported reward) which agents' tasks are selected for execution and among the selected tasks how much optional execution (beyond the mandatory) is allocated to each job of a task.

Since each agent is self-interested, it may report task parameters and values different from the true ones if by doing so its task will be selected to run on the processor. Consider again the example of multimedia video processing; if multiple users are sharing a single processing resource, there may be an incentive for one user to "game" the system to obtain a larger processing time that is disproportionate to the reward that this user derives from the enhanced video quality. The challenge is to align the individual goals of the self-interested agents with the overall system goals. Such challenge calls for algorithmic mechanism design [6] rather than algorithm design. Mechanism design focuses on designing rules in a competitive environment to achieve specific properties such as *truthfulness* and *efficiency*. The truthfulness property guarantees that the agents cannot manipulate the system by lying, and the efficiency property ensures a maximized system-wide objective.

A mechanism for real-time systems will take the task characteristics from each agent and decide which agents obtain the processor. The mechanism also determines the amount that should be paid by the agents who obtained the processor. We are interested in designing mechanisms that give incentives to the agents to report the true characteristics of their tasks and thus, guarantee an efficient processor allocation. Our proposed mechanisms provide tractable solutions and support truthfulness.

1.1 Our Contributions

We address the problem of reward-based scheduling of periodic tasks upon a single processor in a competitive environment by designing truthful mechanisms. This is an NP-complete problem (as will be shown later). We observe that the reward-based scheduling is a K-minded setting rather than a single-minded setting by nature. We first model the reward-based scheduling problem for K-minded bidders as a multi-unit auction, and specify their bidding language. We then design a truthful exact mechanism that uses a pseudo-polynomial-time dynamic programming algorithm to optimally allocate the single processor to a subset of participating agents. In the absence of computationally tractable optimal algorithms for solving this problem, we then design an approximation algorithm to find a near-optimal allocation. In general, approximation algorithms do not necessarily satisfy the properties required to achieve truthfulness. Our proposed approximation mechanism, called MIR-MAX-REW, is a truthful polynomial-time approximation scheme (PTAS) mechanism that gives incentives to agents to reveal their true valuations for the requested execution times of their tasks. We analyze the properties of the MIR-MAX-REW mechanism and perform extensive experiments. The results show that MIR-MAX-REW determines near optimal allocations while satisfying the truthfulness property.

1.2 Related Work

Traditional real-time task scheduling upon a uniprocessor in non-competitive settings has been studied extensively (e.g., [7], [8], [9]). In these traditional hard-real-time systems, the focus has been primarily upon schedulability: determining *a priori* whether a specified system can meet all deadline constraints. Subsequent research has considered systems where meeting all the deadlines is impossible and the system must decide which tasks to complete by their deadlines and which to abort or permit a deadline miss. For these types of systems where successfully meeting all deadlines is not possible, the goal is typically to optimize a given performance metric. The *imprecise computation* [10] model was an early effort to characterize systems where additional execution may lead to improved results; the goal is to obtain a schedule that minimizes the total system error due to the computation imprecision. The related *Increase-Reward-with-Increase-Service* (IRIS) framework [2] also addressed the setting where the scheduler may have to decide on which task to allocate execution based on the "reward" obtained. Baruah et al. [11] focused on a setting where the system was overloaded and each job had an associated value obtained from meeting its deadline; an online scheduler was designed to maximize the accumulated values of the successfully-executed jobs. Aydin et al. [1] studied *reward-based scheduling* for periodic tasks in which there is a reward associated with each task's execution. Each task is composed of a mandatory and an optional part. The mandatory part must meet the task's deadline, while a non-decreasing reward function is associated with the execution of the optional part. The goal is to find a schedule that maximizes the weighted average reward. Researchers have also considered settings where different tasks have different quality-of-service (QoS) settings. Based upon the available resources, the system must determine which QoS levels to select based on some rewards metrics; the Q-RAM model [3] is an example of such a framework. All of these prior works assume that the task characteristics are publicly known, and none of them considers a competitive setting, in which the task's characteristics are private to the agents and the agents compete for resources.

In non-real-time computer systems, the study of resource allocation under competition has recently received much attention. Nisan and Ronen [12] introduced the technique of algorithmic mechanism design for computational problems in a competitive setting. They addressed the problem of minimization of the makespan of tasks on parallel machines by designing a truthful approximation mechanism for the problem. The field of mechanism design has been applied to several computer science problems such as routing [13] and multicast transmission [14].

As mentioned in the previous section, we model the reward-based scheduling problem for K-minded bidders as a multi-unit auction. In multi-unit auctions, there exists multiple identical items, and agents can bid for several items together, called bundle. In the case of multi-unit auctions with single-minded bidders, each bidder is only interested in a single bundle of items. Lehmann et al. [15] proposed truthful greedy mechanisms for single-minded bidders. Kothari et al. [16] proposed an FPTAS mechanism for the single-good multi-unit allocation problem. Their proposed mechanism is approximately truthful for the special case of marginal decreasing valuations. Mu'Alem and Nisan [17] proposed a truthful mechanism for known single minded bidders, where each agent not only is interested in a single bundle of goods, but also the identity of its bundle is known by the mechanism (i.e., it is not a private information). Briest et al. [18] improved the results of [17], and proposed a truthful fully polynomial time approximation scheme (FPTAS) for single-minded bidders. However, in the case of K-minded bidders, it is shown that FPTAS using VCG payments are not possible unless P=NP [19].

In the general K-minded bidders case, an agent bids for K bundles with the desire of receiving one of them. It is worth noting that, the agent specifies the value of each bundle, where both the bundles and the values are private information. Bartal et al. [20] proposed a truthful mechanism for non-single-minded bidders in a multi-good setting, where there exists the same number of units for each good, and each bidder desires no more than a single unit of each good. Babaioff et al. [21] proposed mechanisms for K-minded bidders, where each bidder is interested in exactly one bundle from a set of bundles. However, they considered that all such bundles have exactly the same value for the bidder which is not the case in our reward-based mechanism. Lavi and Swamy [22] proposed a randomized $\frac{1}{2}$-approximation mechanism for K-minded bidders. However, their proposed mechanism uses a weaker notion of truthfulness, truthful in expectation. Dobzinski and Nisan [19] proposed a deterministic truthful 2-approximation mechanism for multi-unit auctions. In particular, they focused on indivisible units. Dobzinski et al. [23] proposed a truthful in expectation FPTAS mechanism for multi-unit auctions. However, their mechanism is randomized with a weaker truthfulness-in-expectation guarantee, while our proposed mechanisms are deterministic in terms of truthfulness. Vocking [24] proposed a randomized universally-truthful PTAS mechanism for multi-unit auctions. Vocking's mechanism is a probability distribution over a set of deterministic truthful mechanisms while our mechanism is deterministic. The setup for Vocking's mechanism assumes that the valuations are given by black boxes that can be queried by the mechanism (weak value queries), while in our case the valuations are given explicitly as a vector.

For real-time scheduling in a competitive setting, there are very few prior results. Porter [25] studied the problem of online real-time scheduling of jobs on a single processor in a competitive environment. In this work, the private characteristics of the agents consists of release time, job length, deadline, and value. However, this work does not address the traditional recurring task models (e.g., sporadic or periodic tasks) which are commonly found in real-time applications. In addition, he considered only single-minded agents. Recently, we proposed a truthful FPTAS allocation mechanism for sporadic tasks competing for allocation upon a single processor [26]. However, our previous study considers a completely different real-time setting, that only deals with single-minded bidders and does not consider a reward-based scheduling model. Thus, the mechanism design techniques used in [26] cannot be applied in our current study. In this paper, we consider the reward-based scheduling model of periodic tasks in a competitive environment with K-minded agents, by permitting each agent to specify K different execution time and value pairs for its task. In the presence of competition, our proposed truthful allocation algorithms determine for each task whether it is selected for execution and what (if any) optional execution for the task is permitted.

1.3 Organization

The paper is organized as follows. In Section 2, we describe the problem of reward-based scheduling of periodic tasks upon a single processor in a competitive environment. In Section 3, we present our proposed exact mechanism, and in Section 4 we present our proposed approximation mechanism that solve the problem in competitive environments. We characterize the properties of the proposed mechanisms. In Section 5, we evaluate the mechanisms by extensive experiments. In Section 6, we summarize our results and present possible directions for future research.

2 Model

In this section, we discuss the original problem of reward-based scheduling for real-time systems in non-competitive settings. Then, we define the competitive version of the problem.

2.1 Reward-Based Scheduling

In reward-based scheduling, there is a set of N periodic tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_N\}$. Each task τ_i is characterized by a tuple $(e_i^m, e_i^o, p_i, r_i(.))$ where e_i^m is a mandatory execution time, e_i^o is an optional execution time, p_i is a period, and $r_i(.)$ is a reward function. Task τ_i releases a job to the system every p_i time units. Each job needs to be executed for e_i^m units of *mandatory execution* before a new job is released by the task. We assume that the relative deadline of each job is equal to task period (i.e., $d_i = p_i$). A job may execute up to e_i^o units of *optional execution* before its deadline. We denote the j-th job of task τ_i by J_{ij}. If J_{ij} completes its mandatory execution time e_i^m and executes its optional portion for $x_{ij} \geq 0$ units, then it obtains a reward of $r_i(x_{ij})$. It is assumed that r_i is a concave, non-decreasing function over $\mathbb{R}^+$ and that $r(0)$ is finite.

The system aims to complete the mandatory execution parts of all jobs by their deadline and maximize the total reward over all tasks in the system by executing a portion of the optional execution of the jobs. Let $x_{ij} \geq 0$ be the amount of optional execution completed by job J_{ij} for any schedule of the task system over its hyperperiod H (i.e., the least common multiple of $p_1, \ldots, p_N$). The total reward for task τ_i over a hyperperiod is given by:

$$\sum_{j=1}^{H/p_i} r_i(x_{ij}) \tag{1}$$

The above reward is obtained when each job completes its mandatory execution by its deadline. The reward-based scheduling problem consists of finding the values of x_{ij}, $i = 1, \ldots, N$ and $j = 1, \ldots, H/p_i$, that give the maximum total reward.

Aydin et al. [1] showed that if the task system is feasible on a single preemptive uniprocessor with EDF scheduling, then there exists an optional assignment of the x_{ij} values maximizing Equation (1), where each job of any task τ_i executes the same amount of optional execution. We will denote such optimal optional execution for task τ_i by α_i. The problem of finding an allocation of optional executions that maximizes the total reward in non-competitive environments (called MAX-REW-NC) is formulated as the following linear program:

$$\text{Maximize } \sum_{i=1}^{N} r_i(\alpha_i) \tag{2}$$

Subject to:

$$\sum_{i=1}^{N} \frac{e_i^m + \alpha_i}{p_i} \leq 1 \tag{3}$$

$$\alpha_i \leq e_i^o, \forall i = 1, \ldots, N \tag{4}$$

$$0 \leq \alpha_i, \forall i = 1, \ldots, N \tag{5}$$

The objective function (2) represents the total reward obtained by the tasks. Constraint (3) ensures that the sum of the utilization of the selected tasks does not exceed 1, where the utilization of a task is the ratio of its execution requirement to its period. Constraints (4) guarantee that for each task at most the total requested optional execution is allocated. Constraints (5) guarantee that the allocation of each task is at least zero. In a non-competitive setting, the mandatory execution parts of all jobs are scheduled, and the objective is to find the allocation of the optional parts to maximize the reward. Since the allocated optional parts of each job of a task over the periods remain the same, maximizing the reward over one period of the task maximizes the overall reward.

2.2 Competitive Reward-Based Scheduling in the Presence of Overload

In the work by Aydin et al. [1], the authors assumed that the set of tasks is feasible for their mandatory execution time under EDF scheduling. In this paper, we consider a new type of reward-based problem (called MAX-REW) which assumes the presence of overload, i.e., it is not possible to schedule all the tasks on the uniprocessor so that all the jobs execute for their mandatory execution part and meet their deadline. In other words the sum of the ratios e_i^m/p_i is greater than one. This assumption reflects the possibility that many agents are competing for a scarce resource.

We consider that each Agent i specifies, via the task specification τ_i, a maximum of discrete K different choices for the amount of optional execution part $\langle 0, \ldots, e_{i,K-1}^o \rangle$, and a vector of values $\langle v_i^0, \ldots, v_i^{K-1} \rangle$, where v_i^k represents the value of $r_i(e_{i,k}^o)$, the value obtained by executing the mandatory part and the $(k+1)$-th amount of the optional execution. For the purposes of this paper, we will assume that all task parameters (i.e., period, execution times, and values) are integers. Note that the value of $r_i(0)$ is the value obtained by executing only the mandatory part e_i^m (where $e_{i,0}^o = 0$). The choices and preferences of Agent i are as follows:

Choice	(Total execution, Value)
1	(e_i^m, v_i^0)
2	$(e_i^m + e_{i,1}^o, v_i^1)$
	$\ldots$
$k+1$	$(e_i^m + e_{i,k}^o, v_i^k)$
	$\ldots$
K	$(e_i^m + e_{i,K-1}^o, v_i^{K-1})$

where $0 \leq e_{i,k}^o \leq e_{i,K-1}^o$, more specifically $e_{i,k}^o$ are sorted such that $e_{i,k-1}^o \leq e_{i,k}^o \leq e_{i,k+1}^o$, and $e_{i,k}^o \in \mathbb{N}^+$. Therefore, $e_{i,K-1}^o$ is the most optional execution requested by Agent i. As a result, each task τ_i is characterized by a $(K+1)$-tuple $(p_i, \langle (e_i^m, v_i^0), \ldots, (e_i^m + e_{i,K-1}^o, v_i^{K-1}) \rangle)$. We assume that $\forall i, e_i^m + e_{i,K-1}^o \leq p_i$. In addition, Agent i receives no reward if its task is not selected for execution.

The MAX-REW problem is to allocate the single processor to a subset of tasks such that the allocation is feasible and maximizes the total value obtained. MAX-

REW is formally defined as follows:

$$\text{Maximize} \sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \cdot v_i^{k-1} \tag{6}$$

Subject to:

$$\sum_{i=1}^{N}\sum_{k=1}^{K} \frac{y_{ik}(e_i^m + e_{i,k-1}^o)}{p_i} \leq 1 \tag{7}$$

$$\sum_{k=1}^{K} y_{ik} \leq 1, \forall i = 1, \ldots, N \tag{8}$$

$$y_{ik} \in \{0,1\}, \forall i = 1, \ldots, N \text{ and } \forall k = 1, \ldots, K \tag{9}$$

where y_{ik} represent the decision variables such that

$$y_{ik} = \begin{cases} 1 & \text{if the mandatory part and optional} \\ & \quad \text{part } e_{i,k-1}^o \text{ of } \tau_i \text{ is allocated} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

The objective function (6) represents the total value obtained by the agents. Constraints (7) ensure that the allocation of the selected tasks does not exceed the capacity of the processor. Constraints (8) guarantee that for each task at most one of the requested execution time is allocated. Constraints (9) represent the integrality requirement for the decision variables y_{ik}. Note that in the above formulation we are implicitly requiring that for each task all of its jobs execute the same amount of optional execution.

2.3 Example

In order to compute the optimal solution for the MAX-REW problem, we rely upon the agents to report their true characteristics. However, agents can manipulate the system by lying. We present an example in a cloud computing setting to show how a strategic agent can affect the optimal solution by untruthful reporting of its task requirements. This example is inspired by [5]. We consider five users (agents) who want to launch their real-time services, and they submit all the information about the real-time applications to the cloud provider. Each agent submits its desired execution units and the value obtained by receiving its request. The requested bundles (execution units) and the values of the tasks owned by these agents are shown in Table 1. The first column shows the agents id, the second column gives the requested mandatory execution along with its value, and the third column gives the requested mandatory and optional execution along with its value, where the value is the summation of the mandatory value and the reward of the optional execution. We assume that $p_i = 10$ for all i, and $K = 2$ (i.e., each agent submits two bundles of requests). For example, Agent 1 submits two bundles $(2, 3)$ and $(6, 7)$, where in the former it specifies its mandatory execution as 2 units, with a value of 3, and in the latter, it specifies that for 4 units of optional execution (total of 6 units), it receives an extra reward of 7-3 = 4 (for a total reward of 7). Since all these tasks

TABLE 1: Agents' characteristics example

Agent(i)	(e_i^m, v_i^0)	$(e_i^m + e_{i,1}^o, v_i^1)$
1	(2,3)	(6,7)
2	(4,6)	(5,10)
3	(2,5)	(4,6)
4	(3,4)	(5,7)
5	(2,2)	(4,5)

cannot be scheduled to execute on a single processor using the MAX-REW linear program Eqs. (6-9), we want to assign the processor to the agents such that we obtain the maximum reward. If each agent is truthful, MAX-REW integer linear program assigns the processor to the second bundle of Agent 2 (5,10), the first bundle of Agents 3 (2,5), and the first bundle of Agent 4 (3,4), which results in a reward of 10+5+4=19.

We analyze two cases in which an agent tries to manipulate the system. In the first case, a non-winning agent manipulates the system by declaring a higher value hoping to become a winner. We assume Agent 1 lies about the obtained reward of its optional execution and reports its second bundle as $(6, 15)$ while everyone else reports their true requests. The second bundle of Agent 1 (6,15), the first bundle of Agent 3 (2,5), and the first bundle of Agent 5 (2,2) would be selected. However, since Agent 1 values the received bundle at 7 units, the solution gives a suboptimal reward of 7+5+2=14. In the second case, a winning agent lies about its required execution units. Now, assume that all agents report their true characteristics except Agent 2 who lies about its required optional unit and declares 6 units in its second bundle (6,10) instead of its true request (5,10). In this case, the first bundle of Agent 1, (2,3), the first bundle of Agent 3, (2,5), and the second bundle of Agent 2, (6,10) would be selected, resulting in a suboptimal reward of 3+5+10=18. In both cases, the total reward is less than the one obtained when all agents report truthfully their requirements. This example motivates the need for mechanisms that incentivize truthful behavior on the part of the agents.

3 Mechanism Design

In this section, we present the basic concepts of mechanism design, and then we introduce a VCG-based exact mechanism for our model that solves MAX-REW.

The problem of reward-based scheduling of periodic tasks upon a single processor in a competitive environment is NP-complete by a reduction to the multiple-choice knapsack problem. In a multiple-choice knapsack problem, the items are subdivided into N classes to pack in a knapsack, where at most one item must be taken from each class. If we consider the processor as a knapsack, the agents can be divided into N classes. In each class, an agent has K items (i.e., requests), and at most one request of an agent must be selected from its class. Since the multiple-choice knapsack problem is NP-complete [27] the reward-based scheduling problem considered in this paper is also NP-complete. The aim is to select a subset of agents for the knapsack maximizing the total value.

We consider a set of agents $\mathcal{N} = \{1, \ldots, N\}$, where each Agent i owns a sporadic task τ_i and submits its characteristics (e.g., the parameters of its task and their values). For every Agent i, we define its *valuation function* $V_i(\cdot)$ as a step function as follows:

$$V_i(e_i) = \begin{cases} v_i^{K-1} & e_i \geq e_i^m + e_{i,K-1}^o \\ \cdots & \\ v_i^k & e_i^m + e_{i,k}^o \leq e_i < e_i^m + e_{i,k+1}^o \\ \cdots & \\ 0 & e_i < e_i^m \end{cases} \qquad (11)$$

where $V_i(e_i)$ denotes the value derived from obtaining e_i, $e_i \leq p_i$, units of execution time. Thus, $V_i(\cdot)$ is a monotonically increasing function (i.e., if $e \leq e'$, then $V(e) \leq V(e')$).

We formulate the problem as a combinatorial auction where agents have preferences over sets of items, called *bundles*. Without loss of generality, we assume that each agent specifies exactly K bundles. In this case, Agent i can define its preferences over K bundles in the form of $(e_{i,k}, v_i^k)$, where $e_{i,k} = e_i^m + e_{i,k}^o$ is the size of the $(k+1)$-th bundle and v_i^k is its value ($0 \leq k < K$). The goal is to select a subset of agents with their assignment maximizing the total value.

In this settings, agents are not single-minded. A single-minded agent cares about one specific bundle; if it does not receive the bundle, it values it at zero [15]. A *multi-minded* agent specifies a set of bids (bundles), and uses a bidding language to express its valuation function over the set of bids [21], [28]. Note that it is more challenging to design truthful mechanisms for cases in which agents are K-minded.

We consider that $V_i(\cdot)$ is represented as a K-minded bid. Each Agent i specifies a collection of pairs $(e_{i,k}, v_i^k)$, where $e_{i,k}$ is the size of $(k+1)$-th bundle and v_i^k is the maximum price that it is willing to pay for that bundle. Based on all bids $\{(e_{i,0}, v_i^0), \ldots, (e_{i,K-1}, v_i^{K-1})\}$ of Agent i, we define the value $V_i(e_i)$ for an allocation e_i to be $\max_{e_{i,k} \leq e_i} v_i^k$. This corresponds to a special case of XOR bidding language [29]. Bidding languages relate to the representation of bids in combinatorial auctions such that agents can encode their valuation and send it to the auctioneer. Our goal is to design a mechanism maximizing the total reward (in economics called *social welfare*) obtained by the agents $\sum_{i \in \mathcal{N}} V_i(e_i)$, where $\sum_{i \in \mathcal{N}} e_i / p_i \leq 1$.

3.1 Preliminaries

In this subsection, we present the basic concepts of mechanism design.

Definition 1 (Mechanism): A mechanism $\mathcal{M} = (\mathcal{A}, \Pi)$ consists of an allocation function $\mathcal{A}$ and a payment rule Π. The allocation function determines an allocation of the resources to a set of winning agents, and the payment rule determines the amount that each agent must pay.

We aim to design a mechanism which allocates the processor to a subset of agents. Since the agents are selfish, they may declare different characteristics than their actual characteristics. We denote the declared characteristics of Agent i by $\hat{\theta}_i = (\hat{p}_i, \langle(\hat{e}_{i,0}, \hat{v}_i^0), \ldots, (\hat{e}_{i,K-1}, \hat{v}_i^{K-1})\rangle)$ and its actual characteristics by $\theta_i = (p_i, \langle(e_{i,0}, v_i^0), \ldots, (e_{i,K-1}, v_i^{K-1})\rangle)$, $\forall i : i \in \mathcal{N}$. We assume that an agent's period is publicly known and an agent cannot lie about their period (i.e., $\hat{p}_i = p_i$).

A mechanism takes all the declared characteristics of the agents as input, and it computes an allocation. The mechanism motivates the agents to reveal their true characteristics by charging them some payments. We denote by $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$ the vector of declared characteristics of the agents and by $\hat{\boldsymbol{\theta}}_{-i}$ the vector of all declared characteristics except the Agent i's declared characteristics (i.e., $\hat{\boldsymbol{\theta}}_{-i} = (\hat{\theta}_1, \ldots, \hat{\theta}_{i-1}, \hat{\theta}_{i+1}, \ldots, \hat{\theta}_N)$). The allocation algorithm $\mathcal{A}$ gets $\hat{\boldsymbol{\theta}}$, the agents' declared characteristics, as an input, and outputs $\mathcal{A}(\hat{\boldsymbol{\theta}}) \subseteq \mathcal{N}$, the set of winning agents along with their allocated bundles. Therefore, Agent i wins if $i \in \mathcal{A}(\hat{\boldsymbol{\theta}})$. The *reward* obtained by the algorithm is given by the sum of the values of the winning agents. The strategy of an agent is represented by its declared characteristics.

Agent i has a utility function $\mu_i = V_i(e_i) - \pi_i$, where π_i is the payment that Agent i is required to pay to the mechanism based on Π, and e_i is the amount of execution allocated to Agent i. The mechanism attempts to maximize the reward while the selfish agents try to maximize their own utility. We are interested in designing a truthful mechanism where it is always in each agent's best interest to declare the true requirements of its task.

Definition 2 (Truthful Mechanism): A mechanism $\mathcal{M}$ is *truthful* (or incentive compatible) if all agents have incentives to reveal their true characteristics. Formally, if the utility of Agent i by true declaration of its characteristics (θ_i) is μ_i and its utility by a non-true declaration $\hat{\theta}_i$ is $\hat{\mu}_i$, we have always $\mu_i(\theta_i, \hat{\boldsymbol{\theta}}_{-i}) \geq \hat{\mu}_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i})$ for any declaration $\hat{\boldsymbol{\theta}}_{-i}$ of other agents' characteristics.

In other words, a mechanism is truthful if truthful reporting is a dominant strategy for the agents, that is, the agents maximize their utilities by truthful reporting independently of what the other agents are reporting.

In this paper, the agents are multi-parameter, which means that they have more than one piece of private information that can lie about. In one-parameter mechanisms, the private information of each agent consists of only one number, while in multi-parameter mechanisms the private information consists of more than one number [30]. In general, multi-parameter auctions are more challenging to design since they consider multi-dimensional settings where each agent's preference is given by multiple values [18], [30]. In the next subsection, we propose an exact truthful mechanism for the MAX-REW problem.

3.2 Truthful Exact Mechanism

In this subsection, we design a Vickrey-Clarke-Groves (VCG)-based truthful mechanism that solves the MAX-REW problem. A VCG mechanism is a generalization of Vickrey's second price auction [31] proposed by Clark [32] and Groves [33]. If VCG mechanisms are obtained from optimal allocation algorithms, the particular VCG payment rule makes truthful reporting the dominant strategy of the agents [12]. This means that the truthfulness property is guaranteed. Our proposed exact mechanism consists of an optimal allocation function and a VCG-based payment rule. In the following, we present our proposed optimal allocation and the VCG-based payment rule.

In order to design a VCG-based mechanism for MAX-REW, we need to design an allocation algorithm that provides the optimal solution to MAX-REW. We propose a dynamic programming algorithm, called DP-MAX-REW, to find the optimal solution to MAX-REW. DP-MAX-REW is given in Algorithm 1. The DP-MAX-REW algorithm has one input parameter, the vector of agents declared characteristics ($\hat{\boldsymbol{\theta}}$). The algorithm has two output parameters: V^*, the optimal total reward, and $\mathbf{x}^*$, the optimal allocation to the agents.

Let $H = \text{lcm}(p_1, \dots, p_N)$ be the hyperperiod, $E_{max} = \max_{\forall i \in \mathcal{N}}\{\hat{e}_i^m + \hat{e}_{i,K-1}^o\}$, and $V_{max} = \max_{\forall i \in \mathcal{N}}\{\hat{v}_i^{K-1}\}$. It is trivial that an upper bound on the maximum execution that can be allocated in any solution is NE_{max}, and an upper bound on the maximum value that can be achieved in any solution is NV_{max}. DP-MAX-REW solves optimally the problem assuming $x_i \in \mathbb{Z}^+$ according to the following dynamic programming recurrence (Lines 16-21):

$$U(i,e,v) = \begin{cases} \min\limits_{\forall k}\{U(i-1,e,v), U(i-1,e-\hat{e}_{i,k}, v-\hat{v}_i^k) + \dfrac{\hat{e}_{i,k}}{\hat{p}_i}\} \\ \qquad\qquad \text{if}(e > \hat{e}_{i,k} \text{ and } v > \hat{v}_i^k) \\ \min\limits_{\forall k}\{U(i-1,e,v), \dfrac{\hat{e}_{i,k}}{\hat{p}_i}\} \\ \qquad\qquad \text{if}(e = \hat{e}_{i,k} \text{ and } v = \hat{v}_i^k) \\ U(i-1,e,v) \qquad \text{otherwise} \end{cases} \tag{12}$$

where $U(i,e,v)$ denotes the minimum utilization of a subset of agents $\{1, 2, \dots, i\}$ whose total allocation is exactly e units of execution, and total value is exactly v. The recurrence considers several cases, not allocating the bundle to Agent i, and allocating $\hat{e}_{i,k}$ units with value $\hat{v}_i^k$ to Agent i, where k is the index of k-th bundle of Agent i. In the case of allocating a bundle to Agent i, the recurrence considers the value of utilization when allocating each bundle to Agent i and finds the minimum among all. In each step, it checks the minimum utilization of allocating $e - \hat{e}_{i,k}$ units with total value of $v - \hat{v}_i^k$ to $i-1$ agents while it allocates the remaining time to the kth bundle of Agent i. As a result, the value of utilization would be $U(i-1, e-\hat{e}_{i,k}, v-\hat{v}_i^k) + \frac{\hat{e}_{i,k}}{\hat{p}_i}$ (first condition). In the case where there is not enough utilization to be allocated to $i-1$ agents (i.e., $e - \hat{e}_{i,k} = 0$ and $v - \hat{v}_i^k = 0$), the value of utilization is calculated based on allocating only to Agent i which is $\frac{\hat{e}_{i,k}}{\hat{p}_i}$ (second condition). In the case of not allocating the bundle to Agent i, the minimum utilization remains the same as the minimum utilization of $i-1$ agents with total allocation of e units and total value of v (third condition). The maximum value of all feasible cases which has a utilization bounded by 1 gives the optimal reward (Lines 22-23). Once the final value is determined, the algorithm finds $\mathbf{x}^*$, the optimal allocation of the agents by looking backward at $U(i,e,v)$. The DP-MAX-REW algorithm finds the optimal solution to the MAX-REW problem. The proof is trivial employing the principle of optimality. DP-MAX-REW is a pseudo-polynomial time algorithm with time complexity given by $O(KN^3E_{max}V_{max})$.

Algorithm 1 DP-MAX-REW: Exact Allocation Algorithm

```
1:  Input: $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \dots, \hat{\theta}_N)$; vector of characteristics (set of
    bundles and their values)
2:  $E_{max} = \max_{\forall i \in \mathcal{N}}\{\hat{e}_i^m + \hat{e}_{i,K-1}^o\}$
3:  $V_{max} = \max_{\forall i \in \mathcal{N}}\{\hat{v}_i^{K-1}\}$
4:  $V^* = 0$
5:  for all $e = 0, \dots, NE_{max}$ do
6:      $k = 1$
7:      for all $v = 0, \dots, NV_{max}$ do
8:          if ($e = \hat{e}_{1,k}$ and $v = \hat{v}_1^k$) then
9:              $U(1,e,v) = \hat{e}_{1,k}/\hat{p}_1$
10:             $k = k + 1$
11:         else
12:             $U(1,e,v) = \infty$
13: for all $i = 2, \dots, N$ do
14:     for all $e = 0, \dots, NE_{max}$ do
15:         for all $v = 0, \dots, NV_{max}$ do
16:             $U(i,e,v) = U(i-1,e,v)$
17:             for all $k = 1, \dots, K$ do
18:                 if ($e > \hat{e}_{i,k}$ and $v > \hat{v}_i^k$) then
19:                     $U(i,e,v) = \min\{U(i,e,v),$
                            $U(i-1, e-\hat{e}_{i,k}, v-\hat{v}_i^k) + \frac{\hat{e}_{i,k}}{\hat{p}_i}\}$
20:                 else if ($e = \hat{e}_{i,k}$ and $v = \hat{v}_i^k$) then
21:                     $U(i,e,v) = \min\{U(i,e,v), \frac{\hat{e}_{i,k}}{\hat{p}_i}\}$
22:             if $U(i,e,v) \leq 1$ and $V^* < v$ then
23:                 $V^* = v$
24: Find $\mathbf{x}^*$ by looking backward at $U(i,e,v)$
25: Output: $\mathbf{x}^*$; optimal allocation
26: Output: $V^*$; optimal total reward
```

In addition to our proposed optimal allocation algorithm, our truthful exact mechanism consists of a VCG-based payment rule. We define our VCG-based mechanism that solves the MAX-REW problem as follows.

Definition 3 (VCG-MAX-REW mechanism): The VCG-MAX-REW mechanism $\mathcal{M} = (\mathcal{A}, \Pi)$ is a VCG-based mechanism, where $\mathcal{A}$ maximizes the reward based on the allocation algorithm DP-MAX-REW, and the payment of

TABLE 2: Different scenarios for Agent 2's request declaration

Case	$(\hat{e}_2^m + \hat{e}_{2,1}^o, \hat{v}_2^1)$	Scenario	Status	Payment	Utility
I	$(5, 10)$	$\hat{v}_2^1 = v_2^1$, $\hat{e}_2^m + \hat{e}_{2,1}^o = e_2^m + e_{2,1}^o$	Win	7	3
II	$(5, 12)$	$\hat{v}_2^1 > v_2^1$, $\hat{e}_2^m + \hat{e}_{2,1}^o = e_2^m + e_{2,1}^o$	Win	7	3
III	$(5, 9)$	$\hat{v}_2^1 < v_2^1$, $\hat{e}_2^m + \hat{e}_{2,1}^o = e_2^m + e_{2,1}^o$	Win	7	3
IV	$(5, 6)$	$\hat{v}_2^1 < v_2^1$, $\hat{e}_2^m + \hat{e}_{2,1}^o = e_2^m + e_{2,1}^o$	Lose	0	0
V	$(6, 10)$	$\hat{v}_2^1 = v_2^1$, $\hat{e}_2^m + \hat{e}_{2,1}^o > e_2^m + e_{2,1}^o$	Win	8	2
VI	$(9, 10)$	$\hat{v}_2^1 = v_2^1$, $\hat{e}_2^m + \hat{e}_{2,1}^o > e_2^m + e_{2,1}^o$	Lose	0	0

each agent i is defined by:

$$\pi_i = \sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}_{-i})} V_j(x_j) - \sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}), j \neq i} V_j(x_j), \quad (13)$$

where the first part is the optimal reward in the case of removing Agent i from the system; the second part is the sum of all except Agent i's value in the optimal case. The VCG-based payment calculates the payment of each agent based on the requests of the rest of the agents.

Since our proposed mechanism, VCG-MAX-REW, uses an optimal allocation algorithm and VCG-based payments, it is therefore, a truthful mechanism [12]. The mechanism has pseudo-polynomial execution time which becomes prohibitive for large problem instances.

3.3 Example

We consider the same example in the cloud computing setting as in Section 2.3, and show how applying the VCG-MAX-REW mechanism gives incentive to the agents to report their true characteristics. In addition, our proposed mechanism is robust against manipulation by an agent. If every agent reports its true requirements, VCG-MAX-REW assigns the processor to the second bundle of Agent 2, (5,10), the first bundle of Agents 3, (2,5), and the first bundle of Agent 4, (3,4), which results in a reward of 10+5+4=19. If Agent 1 lies and reports its second bundle as $(6, 15)$ and everyone else reports their true values, then it wins. VCG-MAX-REW finds the payment of 12 for Agent 1. Since its actual value is 7, its utility is $7 - 12 < 0$. As a result, it is not beneficial for Agent 1 to manipulate the mechanism by lying about its value.

Now, consider that all agents report their true characteristics except Agent 2. We analyze different scenarios based on its submitted request shown in Table 2. In Case I, Agent 2 submits its true request, VCG-MAX-REW selects it as a winning agent, and finds the payment of 7 for Agent 2 resulting in a utility $10 - 7 = 3$. In Case II, Agent 2 submits a request with a higher value 12. In this case, Agent 2 is still a winner and the mechanism determines the same payment of 7 leading to a utility of $10 - 7 = 3$. In Case III, it submits a request with a lower value 9, which is not less than the payment determined by our mechanism (i.e., 7). Thus, Agent 2 is still winning, and the mechanism determines the same payment for the agent as in Case I. However, if Agent 2 submits a request with a lower value below the payment (e.g., $\hat{v}_2^1 = 6$ in Case IV), it becomes a loser leading to zero utility.

We now investigate scenarios in which Agent 2 requests a different execution time than its actual request. In Case V, it submits a higher execution of 6 units instead of its true request of 5, where the agent requests (6,10) as its second bundle. In this case, Agent 2 still wins the bundle. However, she pays more than she pays in Case I, II, and III. The mechanism finds the payment of 8 resulting in a utility $10-8 = 2$. Thus, its utility decreases. In Case VI, Agent 2 becomes a loser by submitting a higher execution 9. We showed that if an agent submits an untruthful request, it can not increase its utility. As a result, the truthful mechanism gives incentive to the agents to not manipulate the system.

4 Truthful PTAS Mechanism

In this section, we introduce our proposed truthful approximation mechanism that solves the MAX-REW problem. Babaioff et al. [21] showed that the model of "single-value multi-minded agents", where an agent desires several different bundles all for the same value, does not fall into the family of one-parameter domains defined by Archer and Tardos [34]. The authors showed that since the desired bundles are not public information, value monotonicity by itself is no longer sufficient for dominant strategy implementation. In our setting, the agents are not only multi-minded, but also have a general monotone valuation function making value monotonicity not sufficient for dominant strategy implementation. As a result, we rely on a concept of maximal-in-range allocation (defined later in this section) which has been used to obtain truthful mechanisms for this more complicated setting.

Nisan and Ronen [35] showed that a family of allocation algorithms, called *maximal-in-range*, yields truthful VCG-based mechanisms. The definition of the maximal-in-range allocation algorithm is as follows:

Definition 4 (Maximal-in-range allocation algorithm): An allocation $\mathcal{A}$ is *maximal-in-range* (MIR) if there exist a set of allocations R such that for every possible input $V_1, \ldots, V_N$, the algorithm outputs the allocation that maximizes the reward in R. That means, for all input valuations $V_1, \ldots, V_N$, the algorithm outputs $\arg\max_{(e_1,\ldots,e_N) \in R} \sum_{i \in \mathcal{N}} V_i(e_i)$, where e_i is the allocation of agent i.

An allocation $\mathcal{A}$ is MIR if it optimizes the reward over some range R, where R is the range of the algorithm. If $\mathcal{A}$ is a maximal-in-range allocation and Π is the VCG payment then $(\mathcal{A}, \Pi)$ is a truthful mechanism [19]. The challenge addressed by our work is to design a truthful mechanism $(\mathcal{A}, \Pi)$ with an approximation guarantee for real-time reward-based scheduling systems which is the first for such a setting.

The main idea in the design of our proposed approximation algorithm, is finding the best partial allocation to a subset of agents first, and then allocating the remaining resources through dynamic programming based on the rounded requests of the unallocated agents. The partial allocation is used as a seed for the approximate solution and it also allows us to control the solution error. The allocation process consists of two phases, a partial allocation and an allocation of rounded requests of remaining agents. To introduce our proposed approximation algorithm, we first need to define the concept of r-round allocations in which at most r agents are used for the partial allocation and the requests of the remaining agents are rounded.

Definition 5 (r-round allocation): Let $H = \text{lcm}(p_1, \ldots, p_N)$ be the hyperperiod. An allocation $(x_1, \ldots, x_N)$ is r-round if there exists a set B of agents, where $|B| \leq r$, and $h = \sum_{i \in B} \frac{H \cdot x_i}{p_i}$, such that the following two conditions hold:

- For each agent $i \notin B$, its allocation over the hyperperiod is a multiple of $\max(\lfloor \frac{H-h}{(N-r)^2} \rfloor, 1)$, where H is the total processing units of the processor,
- $\sum_{i \notin B} \frac{H \cdot x_i}{p_i} \leq \max(\lfloor \frac{H-h}{(N-r)^2} \rfloor, 1) \cdot (N-r)^2$.

Note that $x_i = \hat{e}_i^m + \hat{e}_{i,k}^o$, where $0 \leq \hat{e}_{i,k}^o \leq \hat{e}_{i,K-1}^o$. The second condition means that $\max(\lfloor \frac{H-h}{(N-r)^2} \rfloor, 1) \cdot (N-r)^2$ is an upper bound on the number of units allocated to agents not in B. The r-round is an allocation that schedules the tasks of at most r agents, taking at most one bundle for each agent along with allocating tasks of the remaining agents using rounding. The rounding allows us to reduce the hyperperiod to equi-sized units, and thus, to obtain a polynomial running time for the algorithm. Furthermore, we have chosen the rounding in such a way that we can show that we will lose only a bounded amount of value (when compared with an optimal allocation mechanism).

Based on r-round allocations, we define a range R of allocations, where R is the set of all the r-round allocations for a fixed r, that is, the set of allocations that schedule tasks of at most r agents, taking at most one bundle for each task along with allocating tasks of the remaining agents using rounding. The algorithm finds the best allocation in R leading to near-optimal solution to the MAX-REW problem.

We define our proposed approximation mechanism that solves the MAX-REW problem as follows:

Definition 6 (PTAS-MAX-REW mechanism): The PTAS-MAX-REW mechanism consists of the allocation algorithm MIR-MAX-REW given in Algorithm 2 and the payment function VCG-PAY given in Algorithm 3.

Our proposed allocation algorithm, MIR-MAX-REW, is given in Algorithm 2. MIR-MAX-REW has two input parameters, the vector of agents declared characteristics

Algorithm 2 MIR-MAX-REW: Approximation Algorithm

```
1: Input: $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$; vector of characteristics (set of
   bundles and their values)
2: Input: $r$
3: $H = \text{lcm}(p_1, \ldots, p_N)$
4: $\hat{V} = 0$
5: $\hat{\mathbf{x}} \leftarrow \mathbf{0}$
6: $\mathcal{B} \leftarrow \{B | B \subseteq \mathcal{N}$ and $|B| \leq r\}$
7: for all $B \in \mathcal{B}$ do
8:     for all allocation $(e_1, \ldots, e_N)$, where $\forall i \notin B, e_i = 0$ and
       $\forall i \in B, \hat{e}_i^m \leq e_i \leq (\hat{e}_i^m + \hat{e}_{i,K-1}^o)$ do
9:         $h \leftarrow 0$
10:        $V \leftarrow 0$
11:        for all $i \in B$ do
12:            $h = h + \frac{H \cdot e_i}{p_i}$
13:            $V = V + V_i(e_i)$
14:            $x_i = e_i$
15:        if $h \leq H$ then
16:            $\{(e_1, \ldots, e_N)$ is feasible$\}$
17:            $\tilde{B} = \mathcal{N} \setminus B$
18:            $q = \max(\lfloor \frac{H-h}{(N-r)^2} \rfloor, 1)$
19:            for all $e = 0, \ldots, (N-r)^2$ do
20:                $Z(0, e) = 0$
21:                $A(0, e) = 0$
22:            for all $i \in \tilde{B}$ indexed by $1, \ldots, N-r$ do
23:                for all $e = 0, \ldots, (N-r)^2$ do
24:                    $Z(i, e) = 0$
25:                    $A(i, e) = 0$
26:                    for all $j = 0, \ldots, e$ do
27:                        $a = jq + A(i-1, e-j)$
28:                        if $a \leq H - h$ and $a = eq$ then
29:                            if $Z(i, e) \leq V_i(\frac{jqp_i}{H}) + Z(i-1, e-j)$ then
30:                                $Z(i, e) = V_i(\frac{jqp_i}{H}) + Z(i-1, e-j)$
31:                                $A(i, e) = a$
32:            $V = V + Z(|\tilde{B}|, (N-r)^2)$
33:            Find $\tilde{\mathbf{x}}$ by looking backward at $Z(|\tilde{B}|, (N-r)^2)$
34:            if $\hat{V} \leq V$ then
35:                $\hat{V} = V$
36:                $\hat{\mathbf{x}} \leftarrow \mathbf{x} + \tilde{\mathbf{x}}$
37: Output: $\hat{\mathbf{x}}$; near-optimal allocation
38: Output: $\hat{V}$; near-optimal total reward
```

$(\hat{\boldsymbol{\theta}})$, an integer r, where $r \leq N$. MIR-MAX-REW has two output parameters: $\hat{V}$, the near-optimal reward and $\hat{\mathbf{x}}$, the near-optimal allocation to the agents. The algorithm finds a set $\mathcal{B}$, where each member of $\mathcal{B}$ is a subset of at most r agents in $\mathcal{N}$ (Line 6). Set $\mathcal{B}$ is the range of the algorithm consisting the set of the allocations with at most r agents, and B is one of the r-round allocations. MIR-MAX-REW checks each set $B \in \mathcal{B}$ of agents such that it finds an allocation for agents in B and an allocation for agents not in B which maximizes the total reward considering feasibility of allocations (Lines 7-35). For each B, the algorithm considers all feasible allocations to agents in B based on all of their K specified bundles for each agent (Line 8). The algorithm finds $h = \sum_{i \in B} \frac{H \cdot e_i}{p_i}$, the total units of processing time of the agents in B, and V the total reward of agents in B (Lines 9-14). For analyzing the feasibility of an allocation to agents in B, MIR-MAX-REW checks $\sum_{i \in B} \frac{H \cdot e_i}{p_i} \leq H$ condition (Line 15).

For each B and its feasible allocation to the agents in B according to their bids, the algorithm splits the remaining $H-h$ units of processing time into at most $(N-r)^2$ equi-sized bundles of size $q=\max(\lfloor\frac{H-h}{(N-r)^2}\rfloor,1)$. MIR-MAX-REW optimally allocates these equi-size bundles among the agents that are not in B, $\tilde{B}$, according to the dynamic programming recurrence (Lines 22-30). We define $Z(i,e)$ and $A(i,e)$ as follows: if there exists an allocation of execution of exactly e equi-sized bundles to the first i agents (i.e., $\{\overline{1,2,\ldots,i}\}$), then $Z(i,e)$ is the maximum value of all such allocations and $A(i,e)$ is the number of units from a hyperperiod that have been allocated to agents $\{1,\ldots,i\}$ (i.e., eq); otherwise, if such an allocation does not exist, both $Z(i,e)$ and $A(i,e)$ are zero. Below is the dynamic programming recurrence:

$$\begin{aligned}Z(i,e) \quad &= \max\nolimits_{j\le e}\{V_i(\tfrac{jqp_i}{H})+Z(i-1,e-j)|\\ &(jq+A(i-1,e-j)\le H-h)\\ &\wedge(jq+A(i-1,e-j)=eq)\}\end{aligned}$$

where $jq+A(i-1,e-j)$ represents the number of units from a hyperperiod that have been allocated to the agents $1,\ldots,i$. In the recurrence, we consider allocations to an agent for the whole hyperperiod while the valuation function of the agents refers to a single period. Please recall that Aydin et al. [1] showed that any allocation of execution to jobs of tasks over a hyperperiod could be converted to an allocation (with the same total reward) where each job of the same task has identical execution. As a result, in considering the value of an allocation of j bundles, the argument of the valuation function is the number of units (i.e., jq) divided by the number of periods (i.e., H/p_i). The recurrence considers several cases, not allocating any bundle to i, and allocating $\frac{jqp_i}{H}$ units to agent i, where $j\le e$. For each case, MIR-MAX-REW checks the feasibility of allocating $\frac{jqp_i}{H}$ units by having a condition on the number of units from a hyperperiod that have been allocated (Lines 26-27). The utilization of these agents is limited by the allocation of agents in B. The maximum among all feasible cases gives the value of $Z(i,e)$. The values determined by the dynamic program in MIR-MAX-REW may not give the actual bundle sizes specified by the agent (due to the rounding). However, $V_i(\cdot)$ is a non-decreasing step function (Eq. 11), and it rounds down to the nearest bundle and obtains the same overall reward. In addition, the dynamic program in the MIR-MAX-REW does not explicitly check if e is larger than $\hat{e}_i^m+\hat{e}_{i,K-1}^o$, due to the fact that the $V_i(\cdot)$ step function does not increase after this value. The maximum reward for all agents in $\tilde{B}$ is determined by $Z(|\tilde{B}|,(N-r)^2)$. The total reward for all agents in $B\cup\tilde{B}=\mathcal{N}$ is the sum of V and $Z(|\tilde{B}|,(N-r)^2)$ (Line 31). The algorithm saves the maximum obtained total reward in $\hat{V}$ (Lines 33-35). Finally, the algorithm outputs the best allocation among all allocations considered which gives the maximum total reward. Note that for MIR-MAX-REW (Lines 22-30) we reformulated the dynamic programming used in DP-MAX-REW to consider the hyperperiod as a dimension of the table. Since we are rounding by q units, we reduce the size of the dimension to polynomial in N and ϵ. However, we use a different formulation in DP-MAX-REW considering NE_{max} for one of the dimensions of the table, which leads to a pseudo-polynomial time complexity for DP-MAX-REW.

Algorithm 3 VCG-PAY: Payment Function

```
1: Input: $\hat{\boldsymbol{\theta}}=(\hat{\theta}_1,\ldots,\hat{\theta}_N)$; vector of characteristics (set of
   bundles and their values)
2: Input: $\hat{V}$; reward
3: Input: $\hat{\mathbf{x}}$; allocation
4: Input: $r$
5: for all $i\in\mathcal{N}$ do
6:     ($\hat{V}'$, $\hat{\mathbf{x}}'$) = MIR-MAX-REW($\hat{\boldsymbol{\theta}}_{-i}$, $r$)
7:     $sum_1=0$
8:     $sum_2=0$
9:     for all $j\in\mathcal{N}, j\neq i$ do
10:        $sum_1=sum_1+V_j(\hat{x}'_j)$
11:        $sum_2=sum_2+V_j(\hat{x}_j)$
12:    $\mathcal{P}_i=sum_1-sum_2$
13: Output: $\mathcal{P}=(\mathcal{P}_1,\mathcal{P}_2,\ldots,\mathcal{P}_N)$
```

The payment function for our PTAS, VCG-PAY (Algorithm 3), has four input parameters, the vector of the Agents' declared characteristics ($\hat{\boldsymbol{\theta}}$), the reward $\hat{V}$, the allocation $\hat{\mathbf{x}}$, and the integer r. The output $\mathcal{P}$ is a payment vector for the agents. VCG-PAY calls MIR-MAX-REW to find the allocation and reward obtained without Agent i's participation (Line 6). Based on the allocation to the agents with and without Agent i's participation, VCG-PAY finds the payment for Agent i, where sum_1 is the sum of all values without Agent i's participation in the mechanism, and sum_2 is the sum of all except Agent i's value in the optimal case (Lines 7-12). The time complexity of VCG-PAY is $O(N)$ times the complexity of the MIR-MAX-REW.

Theorem 1: MIR-MAX-REW is a PTAS.

Proof: To prove that MIR-MAX-REW is a PTAS, we need to show the reward obtained by the algorithm is at least $(1-\epsilon)$ times the optimal, and that the time complexity of the algorithm is polynomial in N, K, where $\epsilon=1/(r+1)$.

First, we show that the time complexity of MIR-MAX-REW is polynomial in N, K, and $\log(E_{max})$. Using the XOR bidding language [29], each bundle can be represented by $\log(E_{max})$ bits. Since for each agent there are at most K bundles, collecting the bids from all agents takes $O(NK\log(E_{max}))$. The running time depends on the partial allocation of agents in B and the dynamic programming. For the partial allocation, there are at most $\binom{N}{r}$ possible selections of sets B, where for each B, at most K^r allocations to agents in B are considered. Note that r is defined by ϵ and thus assumed to be a fixed constant. Then, the allocation for agents not in B is determined by dynamic programming. We assume $\tilde{B}=\mathcal{N}\backslash B$, and renumber the agents such that $\tilde{B}=\{1,\ldots,N-r\}$. The utilization of every agent i in $\tilde{B}$ is scaled; i.e., the dynamic programming jumps over q columns instead of filling one column at a time. Thus, the time complexity of the dynamic programming is $O((N-r)^5)$. Overall the algorithm runs in time polynomial in N, K, and $\log(E_{max})$ for every fixed r.

We now show that the solution is within $(1-\epsilon)$ of the optimal solution, where $\epsilon = 1/(r+1)$. To obtain the tightest theoretical bound of the algorithm, we prove the approximation ratio assuming that the set of agents allocated bundles is the first phase is exactly of size r (i.e., $|B| = r$). Note that the approximation ratio trivially still holds when we permit allocations of less than r agents in the first phase (i.e., $|B| < r$), since if the algorithm returned such a smaller first-phase allocation as the best found, then it must be that its valuation is greater than all the allocations considered with $|B| = r$. Let $\mathbf{x}^*$ be the optimal allocation, and V^* be the corresponding optimal value. Assume that MIR-MAX-REW determines an allocation $\mathbf{x}$ and a value V. Without loss of generality, we consider $V_1(x_1^*) \geq V_2(x_2^*) \geq \ldots \geq V_N(x_N^*)$. In the first step, MIR-MAX-REW optimally allocates bundles to r agents in B. We have $h = \sum_{i\in B} \frac{H \cdot x_i^*}{p_i}$, the total units of execution allocated to agents in B. The second step is allocating the remaining units to the agents who were not selected in the first step. The rounding procedure for the remaining agents increases their utilization from $\frac{e_i}{p_i}$ to $\lceil \frac{e_i \cdot H \cdot (N-r)^2}{p_i \cdot (H-h)} \rceil \cdot \lfloor \frac{H-h}{(N-r)^2} \rfloor / H$. This may lead to an infeasible allocation of $\mathbf{x}^*$ based on the new rounded utilization. If $\frac{H-h}{(N-r)^2} \leq 1$, then q is set to one. In this case, there is no inflation of the bundles required, and the remaining $H-h$ units can be optimally allocated. As a result, each agent $i \notin B$ gets the same allocation as x^*. Otherwise, if $q = \lfloor \frac{H-h}{(N-r)^2} \rfloor$ (i.e., $q \neq 1$), at most $(N-r)\lfloor \frac{H-h}{(N-r)^2} \rfloor \leq \frac{H-h}{N-r}$ units are added by rounding up. This is due to the fact that at most $N-r$ agents will be available for the allocation by dynamic programming.

Now, for the case that $q \neq 1$, consider that the rounding of the bundles in the second step for the remaining $N-r$ (denoted $\tilde{B}$) causes the allocation $\mathbf{x}^*$ to be infeasible. By the above paragraph, the total increase in allocation due to rounding is at most $\frac{H-h}{N-r}$. Thus, for the rounding to cause infeasibility of $\mathbf{x}^*$, the original allocation in $\mathbf{x}^*$ to agents of $\tilde{B}$ must be strictly larger than $(H-h) - \frac{H-h}{N-r} = \frac{(H-h)(N-r-1)}{N-r}$. Furthermore, since there are $N-r$ agents in $\tilde{B}$ and a total allocation is strictly larger than $\frac{(H-h)(N-r-1)}{N-r}$, at least one agent of $\tilde{B}$ must have had an allocation in $\mathbf{x}^*$ strictly greater than $\frac{(H-h)(N-r-1)}{(N-r)^2}$, by the generalized pigeonhole principle. Consider removing such an agent $j \in \tilde{B}$ with largest allocation in $\mathbf{x}^*$ exceeding $\frac{(H-h)(N-r-1)}{(N-r)^2}$; then, since $\mathbf{x}^*$ was feasible, the remaining $(N-r-1)$ agents must have been allocated at most $(H-h) - \frac{(H-h)(N-r-1)}{(N-r)^2}$ units in $\mathbf{x}^*$. However, notice that the increased allocation due to rounding for $(N-r-1)$ agents is at most $(N-r-1) * \lfloor \frac{H-h}{(N-r)^2} \rfloor$. Therefore, the removal of j from the allocation permits the remaining elements of $\tilde{B}$ to be feasible allocated with respect to $\mathbf{x}^*$ and rounding. Due to the optimality of the dynamic programming solution, the second stage will return an allocation with value greater than or equal to the allocation where j is removed from consideration.

Since the allocation with j removed serves as a lower bound on the returned allocation from the algorithm, it is sufficient to derive an upper bound on the value of j and consider the decrease in total returned value due to its removal. Since j is not selected in the first step, it is at most as valuable as the $(r+1)$-th agent in $\mathbf{x}^*$. As a result, the value of j is at most $\frac{V^*}{r+1}$. Therefore, with the exclusion of Agent j from the returned allocation, the total decrease in the value is at most $\frac{V^*}{r+1}$. We have, $V \geq V^*(1 - \frac{1}{r+1})$ which gives the required approximation. Therefore, we have $(1-\epsilon)V^* \leq V \leq V^*$, where $\epsilon = 1/(r+1)$. □

Theorem 2: PTAS-MAX-REW is truthful.

Proof: The PTAS-MAX-REW mechanism consists of the allocation algorithm MIR-MAX-REW and the payment function VCG-PAY. MIR-MAX-REW is maximal-in-range because it outputs the best allocation over the the range of r-round allocations. That is, MIR-MAX-REW checks each set $B \in \mathcal{B}$ of agents such that it finds an allocation for agents in B and an allocation for agents not in B (the equi-sized bundles are allocated optimally among the agents not in B) which maximizes the total reward considering the feasibility of allocations. In other words MIR-MAX-REW finds an optimal r-round allocation and therefore it is maximal-in-range. The payment function VCG-PAY is a VCG-based payment function. Since any maximal-in-range allocation algorithm along with a VCG payment scheme is truthful (by [19]), PTAS-MAX-REW is a truthful mechanism. □

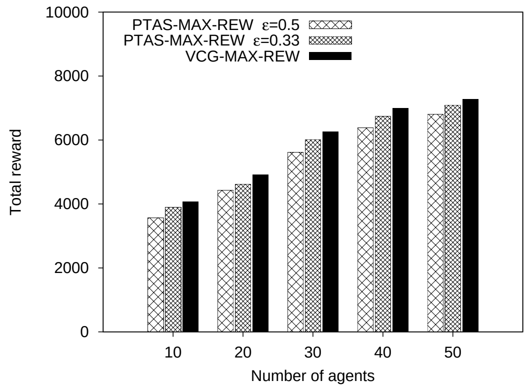
5 Experimental Results

We perform extensive experiments to investigate the properties of VCG-MAX-REW and PTAS-MAX-REW. While it is desirable to compare PTAS-MAX-REW with several other mechanisms, we found out that the existing mechanisms and approaches are not directly comparable to ours and decided to compare it with the optimal mechanism, VCG-MAX-REW. Therefore, we rely on the optimal results obtained by VCG-MAX-REW as a benchmark for our experiments. Both mechanisms are implemented in C++ and the experiments are conducted on a cluster of Intel 2.93GHz Quad Proc Hexa Core nodes with 90GB RAM.

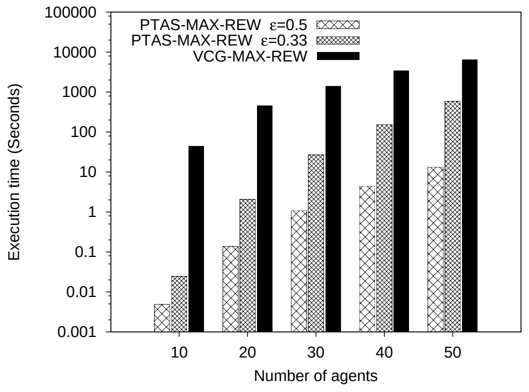
5.1 Experimental Setup

We generate the requested execution units of several problem instances with different number of agents ranging from 10 to 50. For each problem size we generate ten problem instances, and present the average results.

In order to generate the utilizations of the agents, we used the UUniFast-Discard method described in [36] with a discard limit equal to half the number of agents. Since we assume that available resources do not satisfy the demand of all the agents, we should select a target



(a)



(b)

Fig. 1: VCG-MAX-REW and PTAS-MAX-REW performance (K=2): (a) Total reward; (b) Execution time.

utilization greater than 1. In this case, there exists competition among the agents and our mechanisms determine the winners and the payments so that the truthful behavior of agents is guaranteed. Mechanism design is not really required when there is an abundance of resources that can easily satisfy all the agents demands. We set the target utilization U to 5. For the period generation, we use a random uniform number generator to generate a vector of integers within $[10, 50]$. The mandatory execution part is computed by multiplying the corresponding entry in this vector with its associated utilization. We generate $K - 1$ random numbers between mandatory execution and the period as the optional execution units. In the experiments, we use $K = 2, \ldots, 6$, that is, 2-minded to 6-minded agents. We use a random uniform number generator to generate the values as integers within the interval $[1, 1500]$. For PTAS-MAX-REW, we use $\epsilon = 0.50$ and $\epsilon = 0.33$ corresponding to $r = 1$ and $r = 2$, respectively.

5.2 Analysis of Results

We now compare the performance of VCG-MAX-REW and PTAS-MAX-REW for different number of agents. Fig. 1a shows the total reward for 10 to 50 agents obtained by VCG-MAX-REW and PTAS-MAX-REW, where ϵ is 0.50 and 0.33, and $K = 2$. This figure shows that for each number of agents, the total reward increases as ϵ decreases. The PTAS-MAX-REW with both values of ϵ obtains solutions very close to the optimal solutions obtained by VCG-MAX-REW. Fig. 1b shows the execution time of the proposed mechanisms for different number of agents. Note that the vertical axis is in logarithmic scale. This figure shows that by increasing the number of agents, the execution time of all mechanisms increases. In addition, by decreasing ϵ (i.e., increasing the accuracy), the execution time of PTAS-MAX-REW increases. This is the case for any PTAS algorithm. Increasing ϵ from 0.33 to 0.5 results in a very small decrease in the total reward, showing that the solutions obtained by

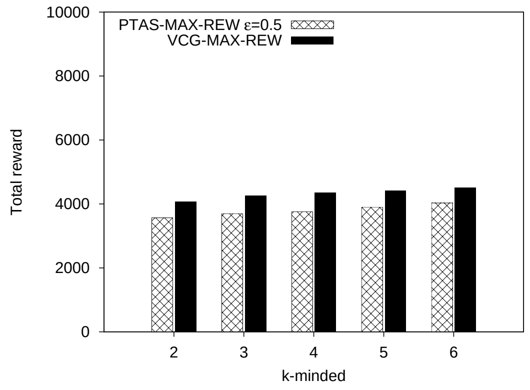


Fig. 2: VCG-MAX-REW and PTAS-MAX-REW total reward for various values of K (N=10)

PTAS-MAX-REW are not very sensitive to ϵ. PTAS-MAX-REW with $\epsilon = 0.5$ obtains a total reward very close to the optimal reward obtained by VCG-MAX-REW and requires a significantly smaller (three orders of magnitude smaller) execution time than VCG-MAX-REW. As an example for $N = 50$ agents, PTAS-MAX-REW obtains a total reward of 6805 requiring an execution time of 13.23 seconds, while VCG-MAX-REW obtains a total reward of 7274 requiring an execution time of 6505.71 seconds. Because PTAS-MAX-REW with $\epsilon = 0.5$ produces near-optimal solutions in a very small amount of time, we recommend it for solving the reward-based scheduling problem in competitive real-time environments.

Figs. 2 and 3 show the effects of considering K-minded agents for different values of K, for the case of 10 agents. In this set of experiments, we allow each agent to have 1 to 5 choices of optional execution in addition to its mandatory execution part making the agent 2-minded to 6-minded. The results show that the total reward slightly increases since there are more configurations with relatively higher reward to choose from by increase in the choices of optional execution. In addition, we analyze the effects of V_{max} on the execution time of the VCG-MAX-REW and PTAS-MAX-REW. Note

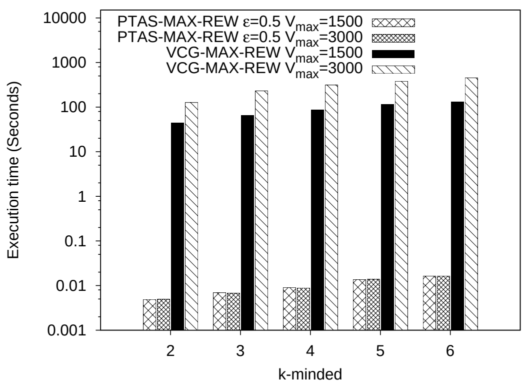


Fig. 3: VCG-MAX-REW and PTAS-MAX-REW execution time for various values of K (N=10)

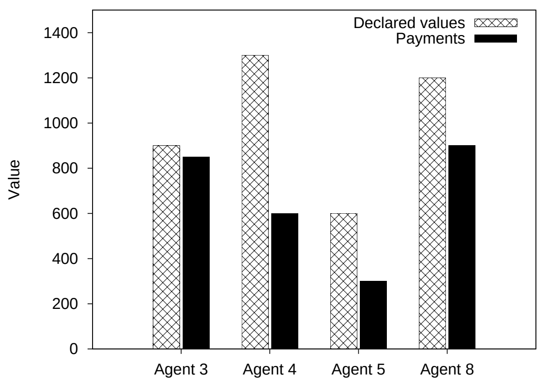


Fig. 4: PTAS-MAX-REW Payments (10 agents and K=2)

that the vertical axis of Fig. 3 is in logarithmic scale. Fig. 3 shows that if V_{max} is within the range $[1, 3000]$ the execution time of VCG-MAX-REW increases compared to the case when V_{max} is within the range $[1, 1500]$, while the execution time of the PTAS-MAX-REW remains the same. As we mentioned, the time complexity of the VCG-MAX-REW is $O(KN^3E_{max}V_{max})$, where V_{max} is the maximum of the agents' declared values, and E_{max} is the maximum of the agents' requested execution units. Thus, the execution time of the VCG-MAX-REW mechanism is highly dependent on V_{max} and E_{max}, where PTAS-MAX-REW's execution time does not depend at all upon these parameters. Therefore, PTAS-MAX-REW is most appropriate for settings where the VCG-MAX-REW's running time is high due to large values of V_{max} and/or E_{max} (e.g., in systems that have both fine-grained execution-time granularity and large task execution requirements).

We consider one instance of the problem with ten 2-minded agents, where Agent 3, 4, 5, and 8 are selected with second, second, first, and second of their corresponding bundles, respectively. The declared values of these agents are shown in Fig. 4. The total reward of these agents is 4000. The payment of these agents are shown in Fig. 4, where the rest of the agents pay zero. The difference between the declared values of the agents and their payments gives their utility. This figure shows that all agents have non-negative utility, and thus, none of the agents lose by participating in the mechanism. The payments of the agents who do not obtain an allocation are zero and are not shown in the figure.

From all above results, we can conclude that our proposed PTAS mechanism obtains near-optimal results in reasonable amount of time while giving incentive to agents to reveal their true valuations. Based on the properties of PTAS-MAX-REW, the agents do not need to strategize since they will not be able to do better than reporting their true valuations.

6 Conclusion

We proposed truthful exact and approximation mechanisms for the reward-based scheduling problem in a competitive environment. Each task obtains a value if the processor successfully schedules all its mandatory jobs, and also an additional reward value if the processor successfully schedules a part of its optional jobs. Each task is owned by a selfish agent, and the agent may try to manipulate the mechanism to obtain an unfair allocation on the processor. We formulated the problem as a multi-parameter combinatorial auction with K-minded agents, where agents can choose K different optional execution units. The proposed mechanisms give incentives to the agents such that it is always in the agent's best interest to report the true characteristics of their tasks. Since the exact mechanism is computationally intractable, we designed a truthful PTAS mechanism. We investigated the properties of our proposed mechanisms by performing experiments. The results showed that the proposed approximation mechanism determines near optimal allocations while giving the agents incentives to report their true characteristics of their tasks.

For future work, we plan to extend this study to multiprocessor systems where there are multiple levels of competition (e.g., the setting where users are competing for allocation in a shared service and the service itself is executing on a shared processing platform). Many online services which utilize a cloud-based computing infrastructure (e.g., Netflix) use such an architecture.

Acknowledgment

We would like to thank the Associate Editor and the reviewers for their helpful and constructive suggestions, which considerably improved the quality of the paper. This research was supported in part by the NSF grants CNS-0953583 and CNS-1116787.

References

[1] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Optimal reward-based scheduling for periodic real-time tasks," *IEEE Trans. Comput.*, vol. 50, no. 2, pp. 111–130, February 2001.

[2] J. Dey, J. Kurose, D. Towsley, C. Krishna, and M. Girkar, "Efficient on-line processor scheduling for a class of iris real-time tasks," in *Proc. of the 13th ACM SIGMETRICS Conference*, 1993.

[3] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A resource allocation model for qos management," in *Proc. 18th IEEE Real-Time Systems Symp.*, 1997.

[4] K. H. Kim, “Reward-based allocation of cluster and grid resources for imprecise computation model-based applications,” *Int. J. of Web and Grid Services*, vol. 9, no. 2, pp. 146–171, 2013.

[5] K. H. Kim, A. Beloglazov, and R. Buyya, “Power-aware provisioning of virtual machines for real-time cloud services,” *Concurrency Comput.: Practice Experience*, vol. 23, no. 13, pp. 1491–1505, 2011.

[6] A. Mas-Colell, M. D. Whinston, and J. R. Green, *Microeconomic Theory*. Oxford University Press, Jun. 1995.

[7] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *J. ACM*, vol. 20, pp. 46–61, January 1973.

[8] A. K. Mok, “Fundamental design problems of distributed systems for hard real-time environments,” Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, Mass., 1983.

[9] J. Lehoczky, L. Sha, and Y. Ding, “The rate monotonic scheduling algorithm: exact characterization and average case behavior,” in *Proc. of Real Time Systems Symposium*, 1989, pp. 166 –171.

[10] K. Lin, J. Liu, and S. Natarajan, “Scheduling real-time, periodic jobs using imprecise results,” in *Proc. of the Real-Time Systems Symposium*, San Fransisco, CA, December 1987.

[11] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha, “On-line scheduling in the presence of overload,” in *Proc. 32nd Symp. on Foundations of Comput. Sci.*, 1991, pp. 100–110.

[12] N. Nisan and A. Ronen, “Algorithmic mechanism design,” *Games and Economic Behavior*, vol. 35, pp. 166–196, 2001.

[13] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker, “A bgp-based mechanism for lowest-cost routing,” in *Proc. 21st ACM Symp. on Principles of Distributed Comp.*, 2002, pp. 173–182.

[14] J. Feigenbaum, C. H. Papadimitriou, and S. Shenker, “Sharing the cost of multicast transmissions,” *Journal of Computer and System Sciences*, vol. 63, no. 1, pp. 21 – 41, 2001.

[15] D. Lehmann, L. Oćallaghan, and Y. Shoham, “Truth revelation in approximately efficient combinatorial auctions,” *Journal of the ACM*, vol. 49, no. 5, pp. 577–602, 2002.

[16] A. Kothari, D. C. Parkes, and S. Suri, “Approximately-strategyproof and tractable multiunit auctions,” *Decision Support Systems*, vol. 39, no. 1, pp. 105–121, 2005.

[17] A. Mu’Alem and N. Nisan, “Truthful approximation mechanisms for restricted combinatorial auctions,” *Games and Economic Behavior*, vol. 64, no. 2, pp. 612–631, 2008.

[18] P. Briest, P. Krysta, and B. Vöcking, “Approximation techniques for utilitarian mechanism design,” *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1587–1622, 2011.

[19] S. Dobzinski and N. Nisan, “Mechanisms for multi-unit auctions,” *J. Artificial Intelligence Res.*, vol. 37, pp. 85–98, 2010.

[20] Y. Bartal, R. Gonen, and N. Nisan, “Incentive compatible multi unit combinatorial auctions,” in *Proc. of the 9th Conf. on Theoretical aspects of rationality and knowledge*, 2003, pp. 72–87.

[21] M. Babaioff, R. Lavi, and E. Pavlov, “Single-value combinatorial auctions and algorithmic implementation in undominated strategies,” *Journal of the ACM*, vol. 56, no. 1, p. 4, 2009.

[22] R. Lavi and C. Swamy, “Truthful and near-optimal mechanism design via linear programming,” *J. ACM*, vol. 58, no. 6, p. 25, 2011.

[23] S. Dobzinski, N. Nisan, and M. Schapira, “Truthful randomized mechanisms for combinatorial auctions,” *Journal of Computer and System Sciences*, vol. 78, no. 1, pp. 15–25, 2012.

[24] B. Vöcking, “A universally-truthful approximation scheme for multi-unit auctions,” *Games and Economic Behavior*, 2013.

[25] R. Porter, “Mechanism design for online real-time scheduling,” in *Proc. 5th ACM Conf. on Electronic Commerce*, 2004, pp. 61–70.

[26] A. Mohammadi, N. Fisher, and D. Grosu, “Truthful mechanisms for allocating a single processor to sporadic tasks in competitive real-time environments,” *IEEE Trans. Comput.*, vol. 63, no. 8, pp. 2066–2079, 2014.

[27] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, 2004.

[28] S. Dobzinski and S. Dughmi, “On the power of randomization in algorithmic mechanism design,” in *Proc. of the 50th Annual IEEE Symposium on Foundations of Computer Science*, 2009, pp. 505–514.

[29] N. Nisan, “Bidding languages,” in *Combinatorial auctions*, P. Cramton, Y. Shoham, and R. Steinberg, Eds. Cambridge: MIT, 2006.

[30] S. Chawla, J. D. Hartline, D. L. Malec, and B. Sivan, “Multi-parameter mechanism design and sequential posted pricing,” in *Proc. 42nd ACM Symp. Theory of Comput.*, 2010, pp. 311–320.

[31] W. Vickrey, “Counterspeculation, auctions, and competitive sealed tenders,” *The Journal of Finance*, vol. 16, no. 1, pp. 8–37, 1961.

[32] E. Clarke, “Multipart pricing of public goods,” *Public choice*, vol. 11, no. 1, pp. 17–33, 1971.

[33] T. Groves, “Incentives in teams,” *Econometrica: Journal of the Econometric Society*, vol. 41, no. 4, pp. 617–631, 1973.

[34] A. Archer and É. Tardos, “Truthful mechanisms for one-parameter agents,” in *Proc. 42nd IEEE Symp. Foundations of Comput. Sci.*, 2001, pp. 482–491.

[35] N. Nisan and A. Ronen, “Computationally feasible vcg mechanisms,” *J. of Artificial Intell. Res.*, vol. 29, no. 1, pp. 19–47, 2007.

[36] R. Davis and A. Burns, “Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems,” in *Proc. 30th IEEE Real-Time Systems Symp.*, Dec. 2009, pp. 398–409.

Lena Mashayekhy received her BSc degree in computer engineering-software from Iran University of Science and Technology, and her MSc degree from the University of Isfahan. She is currently a PhD candidate in computer science at Wayne State University, Detroit, Michigan. She has published more than twenty peer-reviewed papers in venues such as IEEE Transactions on Parallel and Distributed Systems, IEEE BigData, IEEE CLOUD, and ICPP. Her research interests include distributed systems, cloud computing, big data analytics, game theory and optimization. She is a student member of the ACM, the IEEE, and the IEEE Computer Society.

Nathan Fisher received the BS degree from the University of Minnesota, Minneapolis, in 1999, the MS degree from Columbia University, New York, in 2002, and the PhD degree from the University of North Carolina, Chapel Hill, in 2007, all in computer science. He is an associate professor with the Department of Computer Science, Wayne State University, Detroit, Michigan. His research interests are in real-time and embedded computer systems, parallel and distributed algorithms, resource allocation, and approximation algorithms. His current research focus is on multiprocessor scheduling theory and composability of real-time applications. He is a member of the ACM, the IEEE, and the IEEE Computer Society.

Daniel Grosu received the Diploma in engineering (automatic control and industrial informatics) from the Technical University of Iaşi, Romania, in 1994 and the MSc and PhD degrees in computer science from the University of Texas at San Antonio in 2002 and 2003, respectively. Currently, he is an associate professor in the Department of Computer Science, Wayne State University, Detroit. His research interests include parallel and distributed systems, cloud computing, parallel algorithms, resource allocation, computer security, and topics at the border of computer science, game theory and economics. He has published more than ninety peer-reviewed papers in the above areas. He has served on the program and steering committees of several international meetings in parallel and distributed computing. He is a senior member of the ACM, the IEEE, and the IEEE Computer Society.