

Selfish Multi-User Task Scheduling

Thomas E. Carroll and Daniel Grosu*
Dept. of Computer Science
Wayne State University
5143 Cass Avenue
Detroit, Michigan 48202 USA
Email: {tec, dgrosu}@cs.wayne.edu

Abstract

In this paper we formulate and study a new scheduling problem called Selfish Multi-User Task Scheduling. This problem assumes that there are several users, each of them having multiple tasks that need processing on a set of parallel identical machines. Each user is selfish and her goal is to minimize the makespan of her own tasks. We model this problem as a non-cooperative, extensive-form game. We use the subgame perfect equilibrium solution concept to analyze the game which provides insight into the problem's properties. We compute the price of anarchy to quantify the costs due to lack of coordination among the users.

1. Introduction

Scheduling is one of the most studied topics in distributed systems. Inefficient scheduling results in underutilized resources and suboptimal performance. Scheduling has traditionally been implemented in terms of a *coordinator* that maximizes (or minimizes) the system's objective function (e.g., minimize the system makespan). In this model, tasks are submitted to the coordinator who decides when and where they will be processed. One direction of recent research considers abolishing the coordinator and instead, have the schedule generation distributed among the selfish participants. The objective of these selfish participants is to maximize their own welfare. Scheduling involving selfish participants is called *selfish scheduling*.

Scheduling involving *selfish machines* [5, 14, 18] is one model of selfish scheduling. The machines (or more appropriately the organizations that own the machines) drive to maximize their own welfare. Another model of selfish scheduling involves *selfish tasks* [2]. In this model each

task is considered an agent whose objective is to minimize its completion time.

In this paper we expand on selfish scheduling by proposing the *Selfish Multi-User Task Scheduling* (SMTS) model. In SMTS, each user (agent) has several tasks requiring processing. The user's objective is to minimize the completion time of her task set. The users interact to create a schedule that processes all their tasks. Since there is no *a priori* motivation for cooperation, the users' objectives are in conflict, and the schedule may not efficiently share the machines. We quantify the users' welfare as a negative function of the completion time of the user's task set. We interpret this as the more time it takes to complete a user's task set, the greater the cost incurred by the user. We formulate SMTS as a non-cooperative, extensive-form game played among the users. The game consists of turns that alternate between the users. During her turn, the user assigns a job to one of the machines of the distributed system. The *extensive-form game* models the sequential structure of decision making which is inherent in SMTS. SMTS can be viewed as a game with perfect information. *Perfect information* arises when all users are able to compute the welfare for all the others. Some of the schedules will be *subgame perfect equilibria*, which are steady states in that no user can unilaterally diverge and increase its welfare. A user will readily remain in these states as all other actions reduce her welfare.

In this paper we examine the properties of SMTS. We characterize the subgame perfect equilibria for these games. In the special case of SMTS with tasks of identical processing time requirements, we suggest a user strategy that yields a schedule that is a subgame perfect equilibrium. Additionally, we look at the price of anarchy for all schedules that are subgame perfect. The *price of anarchy* [16] quantifies the decrease in efficiency due to the lack of coordination among the decision makers in a resource allocation game.

Related work. Scheduling algorithms in the context of parallel and distributed systems are well studied. A survey on

*Corresponding author.

the subject is [4]. The standard scheduling problem notation originates from Graham *et al.* [13]. We focus specifically on the problem $P||C_{\max}$, which is the minimization of the system makespan. The problem of scheduling on a single machine ($1||C_{\max}$) is trivial. For more than one machine, the problem is solved in polynomial time when preemption is considered ($P|pmtn|C_{\max}$) [17]. In the non-preemptive case, the problem is NP-complete [10]. There exists several approximation algorithms for solving this problem. *List Scheduling (LS)* is a $(2 - 1/m)$ -approximation algorithm (where m is the number of machines) proposed by Graham [11]. Later, Graham devised the improved $(4/3 - 1/3m)$ -approximation algorithm called *Longest Process Time (LPT)* [12]. *Shortest Process Time (SPT)* [21] algorithm reverses the order of LPT. It is characterized as a $(2 - 1/m)$ -approximation algorithm. The *Rounding and Dynamic Programming (RDP)* [15] scheduling algorithm is a $(1 + \epsilon)$ -approximation algorithm (for any fixed $\epsilon > 0$).

Game theory has been applied to areas of computer science where agents are competing to obtain a scarce resource. Scheduling with selfish machines (the agents) was first analyzed by Nisan and Ronen [18], who developed the n -approximation *MinWork* mechanism. Archer and Tardos [3] improved the result by developing a 3-approximation randomized mechanism that is *truthful* (dominant strategy) in expectation only. Auletta *et al.* [5] proposed a deterministic truthful mechanism that is a $(4 + \epsilon)$ -approximation for *fixed* number of machines. Andelman *et al.* [1] improved the result by proposing a truthful $(4 + \epsilon)$ -approximation mechanism for an *arbitrary* number of machines. Continuing with the theme of selfish machines, Carroll and Grosu [6] developed a distributed version of *MinWork* that protects the privacy of the bids. They have also devised truthful mechanisms for *Divisible Load Scheduling* [7, 8, 14]. Angel *et al.* [2] considered a different approach by investigating *selfish tasks* and they devised a truthful $(2 - 1/m)$ -approximate mechanism. A closely related topic is *congestion games* proposed by Rosenthal [20]. Koutsoupias and Papadimitriou [16] studied a congestion game in the context of selfish routing. *Selfish routing* is the problem of routing n users on m parallel links. This led them to propose the *price of anarchy* (also called the *coordination ratio*) which quantifies the cost due to lack of coordination. Gairing *et al.* [9] expanded on selfish routing by modeling it as a game *with incomplete information*.

Our contributions. We propose a new scheduling problem in which selfish users interact to create a schedule. We call this problem *Selfish Multi-User Task Scheduling (SMTS)*. We model it as a non-cooperative, extensive-form game played among the users. We analyze the game by finding its subgame perfect equilibria which we then characterize. Finally, we compute the price of anarchy to study the costs

associated with the lack of coordination among the users.

Organization. The paper is structured as follows. In Section 2 we formulate the *Selfish Multi-User Task Scheduling (SMTS)* problem using game theory and scheduling fundamentals. In Section 3 we present examples of *SMTS*. In Section 4 we formalize the observations made in the previous section. In Section 5 we draw conclusions and present future directions.

2. Problem Formulation

In this section we set forth to formally describe the *Selfish Multi-User Scheduling (SMTS)* problem. It assumes that u users are sharing m *parallel* (identical) machines. User U_i ($i = 1, \dots, u$) has a set $\mathcal{T}_i = \{T_{i,1}, \dots, T_{i,\ell_i}\}$ of ℓ_i tasks that she wants processed. The task $T_{i,j}$ requires $p_{i,j}$ units of time in order to be processed on any one of the machines. The users interact in order to generate a schedule for the system which processes all tasks. In *SMTS*, the users are *strategic* (self-interested and welfare-maximizing): their goal is to minimize their *own* makespan which may occur at the expense of the others and the system. We denote by $S = \{S_1, \dots, S_u\}$ the *schedule* comprising the scheduling strategy S_i of user U_i . The (*scheduling*) *strategy* S_i defines what action U_i will choose for all possible scenarios that she may encounter. The makespan of U_i induced by the schedule S is denoted as

$$C_{U_i} = \max_j C_{i,j}(S), \quad (1)$$

where $C_{i,j}(S)$ is the completion time of task $T_{i,j}$ induced by the schedule S . We denote by $P||C_{\text{user}}$ the *Selfish Multi-User Task Scheduling* problem. C_{user} indicates the fact that the objective of each user is to minimize her own makespan.

Definition 1 ($P||C_{\text{user}}$) *The scheduling problem $P||C_{\text{user}}$ is defined as follows. Given a set of m parallel machines and a set of u users, where each user has a set of tasks \mathcal{T}_i requiring processing, find a schedule $S^* = \{S_1^*, \dots, S_u^*\}$ such that for every user U_i ($i = 1, \dots, u$),*

$$S_i^* \in \arg \min_{S_i} C_{U_i}(\{S_1^*, \dots, S_i, \dots, S_u^*\}). \quad (2)$$

The problem $P||C_{\text{user}}$ is related to $P||C_{\max}$, which is the problem of minimizing the system makespan.

Definition 2 ($P||C_{\max}$) *The scheduling problem $P||C_{\max}$ is defined as follows. Given a set of m parallel machines and a set of tasks $\mathcal{T} = \mathcal{T}_1 \cup \dots \cup \mathcal{T}_u$, find a schedule S^* such that*

$$S^* \in \arg \min_S \max_i C_{U_i}(S). \quad (3)$$

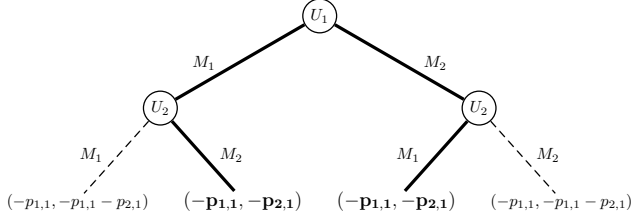


Figure 1. A decision tree modeling a simple SMTS game. The game comprises two machines and two users and each user has a single task. Tasks $T_{1,1}$ and $T_{1,2}$ require $p_{1,1}$ and $p_{2,1}$ units of time to process, respectively.

We note from the above definition that the system makespan is equal to the largest user's makespan.

We formulate $P||C_{\text{user}}$ as a non-cooperative, extensive-form game played among the users. The game consists of $n = \sum_i \ell_i$ turns, which alternate between the users. During her turn, U_i assigns a task to a machine. The subsequent users observe the choice and then make a decision to optimize their assignments.

Definition 3 (SMTS Game) *The SMTS game consists of u selfish users and m parallel machines. Each user U_i has a set \mathcal{T}_i of tasks that requires processing. The game consists of turns that alternate between users. A user assigns a task to a machine during her turn. All users can observe the machine state and thus, can optimize their assignments. The game completes when all tasks are assigned.*

An *extensive-form game* is a dynamic model that expresses the sequential structure of decision making. The sequential structure is inherent in the assignment made by the users in SMTS. Figure 1 depicts a SMTS game for two machines shared by two users and each user has a single task $T_{i,1}$ requiring $p_{i,1}$ units of time for processing. The game has two turns: user U_1 schedules $T_{1,1}$ which is followed by U_2 assigning $T_{2,1}$. User U_2 does not haphazardly assign her task, but first polls the machine loads and then decides which machine to assign her task. A series of actions is a *sequence*. The game in Figure 1 has six such sequences: (M_1) , (M_2) , (M_1M_1) , (M_1M_2) , (M_2M_1) , and (M_2M_2) . The sequence (M_1M_2) indicates that U_1 assigns her job to M_1 and user U_2 assigns her job to M_2 .

Definition 4 (Sequence) [19] *Let $A = (\alpha_1 \dots \alpha_n)$ be a sequence in which action α_i was taken during turn i .*

A sequence has *histories*. The sequence (M_1M_2) has three *subhistories*: \emptyset , (M_1) , and (M_1M_2) . The histories \emptyset and (M_1) are *proper subhistories* of (M_1M_2) ; subhistory (M_1M_2) is a *terminal history*.

Definition 5 (Histories) [19] *The sequence $A = (\alpha_1 \dots \alpha_n)$ has subhistories $h_0 = \emptyset$ and all sequences $h_i = (\alpha_1 \dots \alpha_m)$ ($1 \leq m \leq n$). The subhistory h_j ($j = 0, \dots, n-1$) (where $h_0 = \emptyset$) is a proper subhistory. A sequence that is not a proper subhistory of another sequence is a terminal history.*

The terminal histories are annotated with preferences. The i -th component of the *preference* tuple indicates the cost incurred by U_i . A user's cost is quantified by a negative function of the completion time of her task set. The more time a user awaits the completion of her tasks, the greater her incurred cost. We specify the cost of schedule S incurred by U_i as $-C_{U_i}(S)$. The *payoff function* $u_i(h)$ gives the preference of user U_i given terminal history h . User U_i prefers terminal history h' to h , if $u_i(h') > u_i(h)$ and is indifferent between histories h' and h if $u_i(h') = u_i(h)$. In the game represented in Figure 1, user U_2 prefers history (M_1M_2) to (M_1M_1) as $u_2(M_1M_2) > u_2(M_1M_1)$ ($u_2(M_1M_2) = -p_{2,1}$ and $u_2(M_1M_1) = -p_{1,1} - p_{2,1}$).

Definition 6 (Preference) *The preference of terminal history h is a tuple of payoffs $u_i(h)$, one for each user U_i .*

The SMTS game is a game with perfect information. The users in a game with *perfect information* know one another's type, which they use to compute preferences. The *type* of U_i is her task set \mathcal{T}_i and the processing time requirements $p_{i,1}, \dots, p_{i,\ell_i}$.

Definition 7 (Extensive-Form Game with Perfect Information) [19] *An extensive-form game with perfect information consists of the following:*

- (i) *The set of players;*
- (ii) *The set of terminal histories;*
- (iii) *A function $P(h)$ that assigns the player to the move after subhistory h ;*
- (iv) *Preferences over the terminal histories.*

We formally define the SMTS game from Figure 1 as follows:

- (i) $\{U_1, U_2\}$ is the set of users;
- (ii) $\{(M_1M_1), (M_1M_2), (M_2M_1), (M_2M_2)\}$ is the set of terminal histories;
- (iii) The function $P(h)$ such that $P(\emptyset) = U_1$ and $P(M_1) = P(M_2) = U_2$;
- (iv) The preferences are $u_1(M_1M_1) = u_1(M_1M_2) = u_1(M_2M_1) = u_1(M_2M_2) = -p_{1,1}$, $u_2(M_1M_1) = u_2(M_2M_2) = -(p_{1,1} + p_{2,1})$, and $u_2(M_1M_2) = u_2(M_2M_1) = -p_{2,1}$.

As was mentioned earlier, the objective of user U_i is to minimize her makespan $C_{U_i}(S)$. User U_i is not isolated; the actions of U_i influence and are influenced by the actions of the others. This is apparent as $C_{U_i}(S)$ is a function of the schedule S and not just a function of her scheduling strategy S_i . The solution concept for extensive form games that we consider in this paper is the subgame perfect equilibrium. This equilibrium is a steady state in which no user can unilaterally increase her welfare.

Definition 8 (Outcome) *The outcome $O(S)$ is the terminal history h that is induced by scheduling profile S . The outcome $O_q(S)$ is the terminal history h with subhistory q induced by scheduling profile S .*

Notation: In the rest of the paper we denote by S_{-i} the set of strategies not including the strategy of user U_i . The scheduling strategy S is represented as $\{S_i, S_{-i}\}$.

Definition 9 (Subgame Perfect Equilibrium of Extensive-Form Games with Perfect Information) [19] *A schedule S is a subgame perfect equilibrium if, for every user U_i and every history h where $P(h) = U_i$,*

$$u_i(O_h(S)) \geq u_i(O_h(\{R_i, S_{-i}\})), \quad (4)$$

for every strategy R_i of U_i ,

where $\{R_i, S_{-i}\}$ is the schedule in which U_i chooses strategy R_i and U_j ($j \neq i$) chooses S_j ; outcome $O_h(S)$ is the terminal history q with subhistory h induced by schedule S ; and $u_i(q)$ is the payoff function of U_i at terminal history q .

By definition, for a given game, the set of subgame perfect equilibria (SPE) is a subset of the Nash equilibria.

Definition 10 (Nash Equilibrium of Extensive-Form Games with Perfect Information) [19] *A schedule S is a Nash equilibrium if, for every user U_i ,*

$$u_i(O(S)) \geq u_i(O(\{R_i, S_{-i}\})), \quad (5)$$

for every strategy R_i of U_i ,

where $\{R_i, S_{-i}\}$ is the schedule in which U_i chooses strategy R_i and U_j ($j \neq i$) chooses S_j ; outcome $O(S)$ is the terminal history q induced by schedule S ; and $u_i(q)$ is the payoff function of U_i at terminal history q .

Unlike the Nash equilibrium, the SPEs are *robust* in that they always generate the best response from *any* history h .

The SPEs are computed using backward induction. An extensive form game comprises a subgame at each and every subhistory of the terminal histories. The *length of a subgame* is the largest number of actions needed to complete a game. By convention, a subgame at a terminal history has length one. The *backward induction* procedure locates the

SPEs of the subgames of length i to compute the SPEs of subgames of length $i + 1$. The actions that are subgame perfect are identified by thick, solid lines in Figure 1. A terminal history composed entirely of thick, solid lines indicates the outcome for a game-wide SPE. Additionally, we emphasize the preferences of these SPE. The game in Figure 1 has two schedules that are SPEs: $\{\{M_1\}, \{M_2M_1\}\}$ and $\{\{M_2\}, \{M_2M_1\}\}$. User U_i strategy indicates her choices made from top to bottom, left to right. The schedule $\{M_1, M_2M_1\}$ specifies that U_1 will assign her job to machine M_1 and U_2 will assign her job to M_2 . The second action of M_1 in strategy (M_2M_1) corresponds to the condition in which U_1 assigns her job to M_2 .

The SPE may not be the socially optimal solution to the problem. The *social welfare* indicates how efficiently the public good is shared. In the context of $P||C_{\text{user}}$, the social welfare is maximized when C_{max} (the makespan of the system) is minimized.

Definition 11 (Social Optimum) *Schedule S^* is the social optimum if it yields the smallest system makespan, i.e., $S^* \in \arg \min_S C_{\text{max}}(S)$.*

The *price of anarchy* [16] quantifies the costs arising from the *lack of coordination* among the decision makers in a resource allocation game.

Definition 12 (Price of Anarchy) [16] *The price of anarchy \mathbb{A} is the worst-case ratio of the social welfare obtained at equilibrium to the welfare obtained at social optimum.*

SMTS is a non-preemptive scheduling problem. The results we present in this paper are based on the scheduling algorithm LS [11].

Definition 13 (List Scheduling (LS)) [11] *The LS non-preemptive scheduling algorithm assigns each task to the least load machine.*

In the next section we discuss the solutions of the SMTS problem and characterize their properties.

3. Selfish Multi-User Scheduling

We employ extensive-form games with perfect information for modeling SMTS. *Perfect information* is satisfied when all users know each others type. The *type* of user U_i consists of her job set \mathcal{T}_i and processing times $p_{i,k} \forall k$. A user determines her next move by deducing for each of her actions what subsequent moves the users, including herself, will take and then choosing the action that leads to the most preferred terminal history. Using backward induction, we find the set of schedules that are *subgame perfect equilibria* (SPEs).

We can characterize the number of SPE associated with a specific instance of a game. The number of SPE are associated with the number of histories where the user whose turn to play is indifferent between actions. A user is indifferent between actions when the actions lead to terminal histories of equal payoff. The *minimum* number of SPE of a SMTS game is m as user U_1 chooses one of m machines to assign her first task (U_1 is indifferent since all the machines have identical processing capacity). Games that have a great number of histories leading to states in which the machines are equally loaded have vastly more SPE than similar sized games without this attribute. This is especially observable when all the tasks to be scheduled require identical processing time. The *maximum* number of SPE will be less than the product of the number of actions available at each subgame.

We now give three examples of SMTS games. In the next section, we draw results from these examples.

Example 1 We examine the simplest game in which all tasks have identical processing requirements, *i.e.*, $p_{i,k} = p \forall i, k$. Figure 2 depicts one such game. The game involves two machines, two users, and two tasks per user. We solve for the SPEs by invoking backward induction. First, we find the SPEs associated with the subgames at the bottom most level of the decision tree. An action is a component of a SPE if the user whose turn it is to move prefers it to the other actions. User U_i prefers $\alpha_1 = M_1$ to $\alpha_2 = M_2$ if assigning her task to M_1 minimizes her makespan C_{U_i} ; she is indifferent if both actions result in the same payoff. The results found in the level j are used to discover the SPEs in level $j - 1$. The process completes when we find the SPEs for the root node of the decision tree. The game depicted in Figure 2 has twentyfour SPEs. One of the SPE is the schedule $S = \{\{M_1 M_2 M_1 M_1 M_1\}, \{M_1 M_1 M_2 M_2 M_2 M_1 M_2 M_1 M_1 M_1\}\}$, where the i -th component is strategy S_i of U_i . The strategy S_i consists of all actions for all subgames from top to bottom, left to right that U_i plays. Studying the twentyfour schedules, we note that user U_i may not assign her task to the least loaded machine. This may be surprising, but it is a consequence of subgame perfection: every user has beliefs about every other user and these beliefs are always correct (A chain of knowledge is created such that “ U_i knows that U_j knows” $\forall i, j$). Schedule S is an example where U_2 does not assign her first task to the least loaded machine. She employs a strategy in which she assigns task $T_{2,1}$ to the machine that will process $T_{1,1}$. Another observation of the game is that all SPEs yield the optimal C_{\max} and therefore, the price of anarchy is $\mathbb{A} = 1$. This property is specific to games of identical tasks as later examples will show. Due to the structure of SMTS games with identical tasks, the system makespan and the makespan of the user who assigned the last task are equal, *i.e.*, $C_{\max} = C_{U_i} = C_{i,k}$, where $T_{i,k}$ is the last task assigned in the game.

The next two examples illustrate the general case of tasks with varying processing time requirements.

Example 2 The general games in which the tasks have different processing time requirements are usually characterized by a fewer number of SPEs. This is due to the smaller number of instances in which the machines are equally loaded. We consider the game depicted in Figure 3 consisting of two machines, two users, and two tasks per user. The tasks have processing times $p_{1,1} = 2$, $p_{1,2} = 1$, $p_{2,1} = 5$, and $p_{2,2} = 4$. The game has eight SPEs, all of them yielding a system makespan $C_{\max} = 7$. The optimal makespan for this game is $C_{\max} = 6$ (Machine M_1 processes tasks $T_{1,1}$ and $T_{2,2}$; machine M_2 processes $T_{1,2}$ and $T_{2,1}$); thus, the price of anarchy is $\mathbb{A} = 7/6$.

Example 3 The final example is a game consisting of two machines, two users, and three tasks per user. The processing times are $p_{1,1} = 11$, $p_{1,2} = 9$, $p_{1,3} = 7$, $p_{2,1} = 5$, $p_{2,2} = 3$, and $p_{2,3} = 1$. This game is large: it consists of 511 decision nodes and 512 outcomes. This game has only two SPEs. If task $T_{2,1}$ is assigned to the least loaded machine, U_1 assigns her task $T_{1,2}$ to the machine which is processing $T_{1,1}$. This machine at this point in time has six more units of load when compared against the least loaded machine. The SPEs result in a makespan of $C_{\max} = 20$; the optimum is $C_{\max}^* = 18$ (M_1 processes tasks $T_{1,1}$ and $T_{1,3}$; the remaining tasks are processed on M_2). The price of anarchy is $\mathbb{A} = 10/9$.

4. Results

In this section we formalize the properties of SMTS. We first consider SMTS with tasks having identical processing requirements, *i.e.*, $p_{i,k} = p \forall i, k$. In these games, all users assigning their tasks to the least loaded machines yield schedules that are SPEs. We call this strategy SMTS-LS (*Selfish Multi-User Task Scheduling - List Scheduling*).

Definition 14 (SMTS-LS) User U_i employs the SMTS-LS scheduling strategy if she assigns every task $T_{i,k} \in \mathcal{T}_i$ to the least loaded machine.

The load structure of the machines is well defined when all users employ SMTS-LS: users assign their tasks to machines with available start times of s before assigning them to machines with start times $s + p$.

Theorem 1 (SMTS-LS Strategy Yields SPE) The schedule S is a SPE when all users employ the SMTS-LS strategy.

Proof: Let schedule $S = \{S_1, \dots, S_u\}$ be composed of SMTS-LS strategies S_i . The start time of task $T_{i,k}$ is $s_{i,k}$ in schedule S . Denote by $R = \{R_i, S_{-i}\}$ the schedule with user U_i employing strategy R_i and user U_j employing

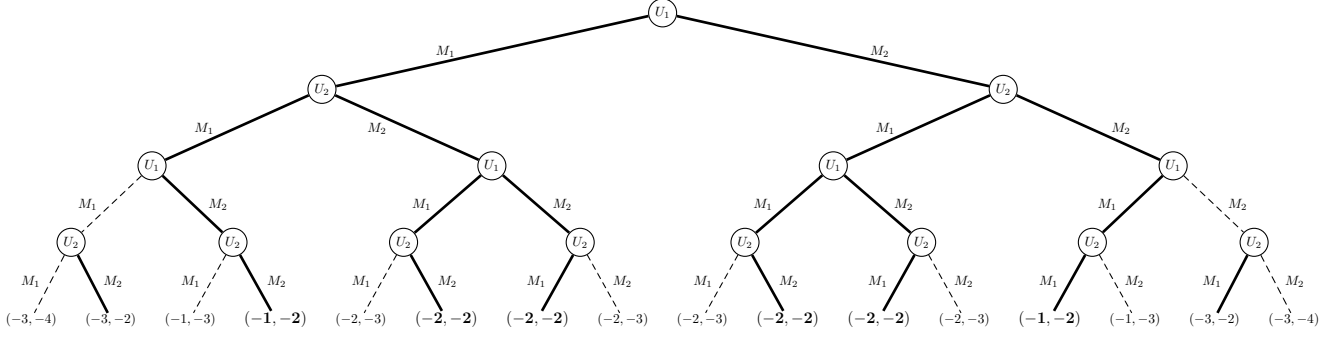


Figure 2. SMTS game with two machines and two users, each user has two tasks. The tasks processing time requirements are $p_{1,1} = p_{1,2} = p_{2,1} = p_{2,2} = p$.

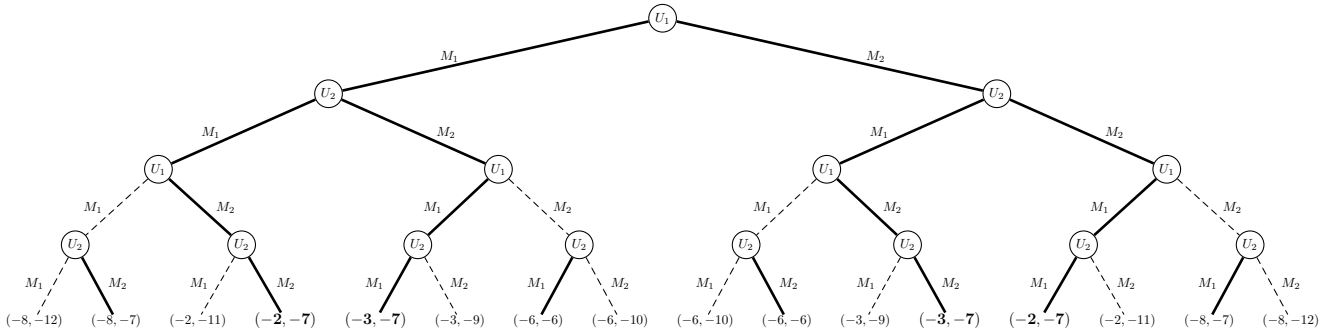


Figure 3. SMTS game with two machines and two users, each user has two tasks. The processing time requirements are $p_{1,1} = 2, p_{1,2} = 1, p_{2,1} = 5,$ and $p_{2,2} = 4$.

the SMTS-LS strategy S_j . If task $T_{i,k}$ is *not* assigned to the least loaded machine, its start time must increase by at least p , i.e., $r_{i,k} \geq s_{i,k} + p$. For a task $T_{i,u}$ that started before $r_{i,k}$ in schedule S , its start time can be earlier due to the time slot that $T_{i,k}$ vacated, i.e., $r_{i,u} \leq s_{i,u}$. For tasks $T_{i,v}$ that started on or after $r_{i,k}$ in schedule S , its start can be delayed due to the slot now occupied by $T_{i,k}$, i.e., $r_{i,v} \geq s_{i,v}$. To show that S is a subgame perfect equilibrium, we must show, for every user U_i and every history h where $P(h) = U_i, u_i(O_h(S)) \geq u_i(O_h(R))$ for every strategy R_i of U_i . Let tasks $T_{i,u}, T_{i,k}$, and $T_{i,v}$ be assigned after history h . If $C_{U_i}(R) \neq C_{i,u}(R), C_{U_i}(R) \neq C_{i,k}(R)$, and $C_{U_i}(R) \neq C_{i,v}(R)$, then there exists a task $T_{i,w}$ in h such that $C_{U_i}(R) = C_{i,w}(R)$ and thus, $C_{U_i}(R) = C_{U_i}(S)$. By definition, the case $C_{U_i}(R) = C_{i,u}(R)$ cannot occur. If $C_{U_i}(R) = C_{i,v}(R)$, then $C_{U_i}(R) \geq C_{U_i}(S)$ as the start of task $T_{i,v}$ may be delayed. If $C_{U_i}(R) = C_{i,k}(R)$, then $C_{U_i}(R) \geq C_{U_i}(S)$ as $r_{i,k} \geq s_{i,k} + p$. It is $C_{U_i}(R) \geq C_{U_i}(S)$ and not $U_i(R) > C_{U_i}(S)$, returning $T_{i,k}$ to the least loaded machines may increase the start time of tasks $T_{i,u}$. ■

The next property we investigate is the social welfare. All SPEs, even those that do not originate from SMTS-

LS, will yield a system makespan equal to the optimal makespan, i.e., $C_{\max}^* = C_{\max}(S^*) \forall S^*$ that are SPE.

Theorem 2 (SPE Outcomes are Optimal) *All SPEs of games with tasks requiring equal processing times yield optimal C_{\max} .*

Proof: We base our proof on the fact that in optimally loaded systems, C_{\max} and the makespans of the least loaded machines differ by at most p units of time. We prove this theorem by contradiction. We assume that schedule S is a SPE that it does not yield an optimal $C_{\max}(S)$. In optimally loaded systems, C_{\max} and the makespans of the least loaded machines differ by at most p units of time. Thus, from our assumption, $C_{\max}(S)$ and the makespan of the least loaded machine differ by more than p . Let task $T_{i,k}$ be such that $C_{i,k}(S) = C_{\max}(S)$. User U_i can unilaterally increase her welfare by transferring the task to the least loaded machine. Thus, S is not a SPE. Since all SPEs have an optimal load structure, they are optima. ■

With the knowledge of the system makespan, we compute the price of anarchy.

Theorem 3 (Price of Anarchy) *All SPEs of SMTS games*

with tasks requiring identical processing times yield a price of anarchy of 1 ($\mathbb{A} = 1$).

Proof: This property follows directly from Theorem 2. All SPEs yield an optimal system makespan C_{\max}^* . Therefore, $\mathbb{A} = C_{\max}^*/C_{\max}^* = 1$. ■

We now consider the general problem of users assigning tasks that have non-identical processing time requirements (This is referred to as *non-identical size tasks*). This problem is much more complex and difficult to analyze. A universal strategy such as SMTS-LS can fail to produce a schedule that is a SPE for these types of games.

Theorem 4 (SMTS-LS and Non-Identical Size Tasks) *The SMTS-LS strategy does not guarantee a schedule that is a SPE for the SMTS problem with non-identical sized tasks.*

Proof: SMTS-LS fails to produce a SPE in the game of Example 3. In each of the SPEs, user U_1 assigns $T_{1,2}$ to the machine executing $T_{1,1}$, which is *not* the least loaded machine. ■

Games with non-identical tasks have greater freedom of task assignments. Users have opportunities to decrease their makespan which is not possible when employing a universal strategy such as SMTS-LS. These opportunities arise as the users are essentially exploring the solution space using backward induction which requires exponential time to run.

Even though a universal strategy does not exist for these games, we can still give an upper bound on the price of anarchy.

Theorem 5 (Price of Anarchy for Non-Identical Size Tasks) *The upper bound on the price of anarchy for SMTS games with non-identical size tasks is 2.*

Proof: Let C_{\max}^* be the optimal makespan of a specific game. Let S be a SPE for the game with the property $C_{i,k}(S) = C_{\max}(S)$. The schedule S cannot yield a makespan twice the optimum, i.e., $C_{\max}(S) \leq 2C_{\max}^*$. The most time all machines can be busy is $C_{pmtn} = \frac{1}{m} \sum p_{j,i} \forall j, l$, which is the makespan obtained when using preemptive (*pmtn*) scheduling. Additionally, it is known that $C_{pmtn} \leq C_{\max}^*(S)$ and $p_{i,k} \leq C_{\max}^*$. If $C_{i,k}(S) > 2C_{\max}^*$, there must be a least loaded machine with its makespan less than C_{\max}^* . Assigning $T_{i,k}$ to this machine will increase the welfare of user U_i and thus, S is not a SPE. Therefore, $C_{\max}(S) \leq 2C_{\max}^*$ and the price of anarchy is $\mathbb{A} \leq 2C_{\max}^*/C_{\max}^* = 2$. ■

5. Conclusions

In this paper we proposed and formulated the Selfish Multi-User Task Scheduling (SMTS) problem. SMTS consists of u selfish users sharing a distributed system of m parallel machines. The objective of a user is to minimize the

completion time of her task set. We formulate this problem as a non-cooperative, extensive form game played among the users. The game is divided into turns; during a turn a user assigns one of her tasks to a machine. We examined the subgame perfect equilibria and then formalized their properties. For the special case of games with tasks having identical processing time requirements, we defined a strategy called SMTS-LS in which users always make assignments to the least loaded machine. When all users employ the SMTS-LS strategy, the resulting schedule is a subgame perfect equilibrium.

We have noticed that the payoffs of SMTS-LS schedules in general games are similar to the payoffs of the SPEs. For future work we are planning to investigate the perfect ϵ -Nash equilibrium solution for SMTS games. A schedule is a *perfect ϵ -Nash equilibria* if, for every user U_i and for every history h where $P(h) = U_i$, $u_i(O_h(\{S\})) + \epsilon \geq u_i(O_h(\{R_i, S_{-i}\}))$, for every strategy R_i of U_i , for a given ϵ , where $\{R_i, S_{-i}\}$ is a schedule with user U_i employing strategy R_i and U_j employing strategy S_j ; $O_h(S)$ is the terminal history with subhistory h induced by the schedule S ; and $u_i(q)$ is the payoff of U_i at the terminal history q . We believe that this concept will allow us to find deterministic scheduling strategies for solving the general SMTS problem.

Acknowledgment. This research was supported in part by Wayne State University Faculty Research Grant, 2005.

References

- [1] N. Andelman, Y. Azar, and M. Sorani. Truthful approximation mechanism for scheduling selfish related machines. In *Proc. of the 22nd International Symp. on Theoretical Aspects of Comp. Sci. (STACS '05)*, pages 69–82, 2005.
- [2] E. Angel, E. Bampis, and F. Pascual. Truthful algorithms for scheduling selfish tasks on parallel machines. In *Proc. of the 1st Workshop on Internet and Network Economics (WINE 2005)*, pages 698–707, Dec. 2005.
- [3] A. Archer and Eva Tardos. Truthful mechanisms for one-parameter agents. In *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science (FOCS'01)*, pages 482–491, Oct. 2001.
- [4] M. J. Atallah, editor. *Algorithms and Theory of Computation Handbook*. CRC Press, 1998.
- [5] V. Auletta, R. D. Prisco, P. Penna, and G. Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. In *Proc. of the 21st International Symp. on Theoretical Aspects of Comp. Sci. (STACS '04)*, pages 608–619, 2004.
- [6] T. E. Carroll and D. Grosu. Distributed algorithmic mechanism design for scheduling on unrelated machines. In *Proc. of the 8th International Symp. on Parallel Architectures, Algorithms, and Networks (ISPAN'05)*, pages 194–199, Dec. 2005.
- [7] T. E. Carroll and D. Grosu. A strategyproof mechanism for scheduling divisible loads in bus networks without control

- processors. In *Proc. of the 8th Workshop on Advances in Parallel and Distributed Computational Models (APDCM 2006)*, Apr. 2006.
- [8] T. E. Carroll and D. Grosu. A strategyproof mechanism for scheduling divisible loads in tree networks. In *Proc. of the 20th IEEE International Parallel and Distributed Processing Symp. (IPDPS 2006)*, Apr. 2006.
- [9] M. Gairing, B. Monien, and K. Tiemann. Selfish routing with incomplete information. In *Proc. of the 17th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'05)*, pages 203–212. ACM Press, July 2005.
- [10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, USA, 1979.
- [11] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Tech. J.*, 45:1563–1581, 1966.
- [12] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17(2):416–429, Mar. 1969.
- [13] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Ronnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Discrete optimization II, Ann. Discrete Math.*, 5:287–326, 1979.
- [14] D. Grosu and T. E. Carroll. A strategyproof mechanism for scheduling divisible loads in distributed systems. In *Proc. of the 4th International Symp. on Parallel and Distributed Computing (ISPD'05)*, pages 83–90. IEEE Computer Society, July 2005.
- [15] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. ACM*, 34(1):144–162, Jan. 1987.
- [16] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. *Lect. Notes in Comp. Sci.*, 1563:404–413, 1999.
- [17] R. McNaughton. Scheduling with deadlines and loss functions. *Management Sci.* 6, pages 1–12, 1959.
- [18] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behaviour*, 35(1/2):166–196, Apr. 2001.
- [19] M. J. Osborne. *An Introduction to Game Theory*. Oxford University Press, New York, NY, USA, 2004.
- [20] R. W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International J. of Game Theory*, 2:65–67, 1973.
- [21] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:56–66, 1956.