# Non-monetary fair scheduling — a cooperative game theory approach

Piotr Skowron
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
Warsaw, Poland
p.skowron@mimuw.edu.pl

Krzysztof Rzadca
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
Warsaw, Poland
krzadca@mimuw.edu.pl

## ABSTRACT

We consider a multi-organizational system in which each organization contributes processors to the global pool but also jobs to be processed on the common resources. The fairness of the scheduling algorithm is essential for the stability and even for the existence of such systems (as organizations may refuse to join an unfair system).

We consider on-line, non-clairvoyant scheduling of sequential jobs. The started jobs cannot be stopped, canceled, preempted, or moved to other processors. We consider identical processors, but most of our results can be extended to related or unrelated processors.

We model the fair scheduling problem as a cooperative game and we use the Shapley value to determine the ideal fair schedule. In contrast to the current literature, we do not use money to assess the relative utilities of jobs. Instead, to calculate the contribution of an organization, we determine how the presence of this organization influences the performance of other organizations. Our approach can be used with arbitrary utility function (e.g., flow time, tardiness, resource utilization), but we argue that the utility function should be strategy resilient. The organizations should be discouraged from splitting, merging or delaying their jobs. We present the unique (to within a multiplicative and additive constants) strategy resilient utility function.

We show that the problem of fair scheduling is NP-hard and hard to approximate. However, for unit-size jobs, we present a fully polynomial-time randomized approximation scheme (FPRAS). We also show that the problem parametrized with the number of organizations is fixed parameter tractable (FPT). In cooperative game theory, the Shapley value is considered in many contexts as "the" fair solution. Our results show that, although the problem for the large number of organizations is computationally hard, this solution concept can be used in scheduling (for instance, as a benchmark for measuring fairness of heuristic algorithms).

## Categories and Subject Descriptors

C.1.4 [**Computer Systems Organization**]: Processor Architectures—*Parallel Architectures*; F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*

## Keywords

fair scheduling, cooperative game theory, Shapley value, cooperation, strategy resistance, approximation algorithms, inapproximability, fairness


*SPAA'13,* June 23–25, 2013, Montréal, Québec, Canada.


## 1. INTRODUCTION

In multi-organizational systems, participating organizations give access to their local resources; in return their loads can be processed on other resources. The examples of such systems include PlanetLab, grids (Grid5000, EGEE), or organizationally distributed storage systems [16]. There are a few incentives for federating into consortia: the possibility of decreasing the costs of management and maintenance (one large system can be managed more efficiently than several smaller ones), but also the willingness to utilize resources more efficiently. Peak loads can be offloaded to remote resources. Moreover, organizations can access specialized resources or the whole platform (which permits e.g. testing on a large scale).

In the multi-organizational and multi-user systems fairness of the resource allocation mechanisms is equally important as its efficiency. Efficiency of BitTorrent depends on users' collaboration, which in turn requires the available download bandwidth to be distributed fairly [32]. Fairness has been also discussed in storage systems [4,14,15,17,40,41,44] and computer networks [42]. In scheduling, for instance, a significant part of the description of Maui [19], perhaps the most common cluster scheduler, focuses on the fair-share mechanism. Nevertheless there is no universal agreement on the meaning of fairness; next, we review approaches most commonly used in literature: distributive fairness and game theory.

In distributive fairness organizations are ensured a fraction of the resources according to *predefined (given) shares*. The share of an organization may depend on the perceived importance of the workload, payments [4,14,15,40]; or calculated to satisfy (predefined) service level agreements [17,21,44]. The literature on distributive fairness describes algorithms distributing resources according to the given shares, but does not describe how the shares should be set. In scheduling, distributive fairness is implemented through fair queuing mechanism: YFQ [1], SFQ and FSFQ [10,20], or their modifications [4,14,15,17,40,41,44,45].

A different approach is to optimize directly the performance (the utility) of users, rather than just the allocated resources. Kostreva el al. [23] proposes an axiomatic characterization of fairness based on multi-objective optimization; Rzadca et al. [34] applies this concept to scheduling in a multi-organizational system. Inoie et al. [18] proposes a similar approach for load balancing: a fair solution must

be Pareto-optimal and the revenues of the players must be proportional to the revenues in Nash equilibrium.

While distributive fairness might be justified in case of centrally-managed systems (e.g. Amazon EC2 or a single HPC center), in our opinion it is inappropriate for consortia (e.g., PlanetLab or non-commercial scientific systems like Grid5000 or EGEE) in which there is no single "owner" and the participating organizations may take actions (e.g. rescheduling jobs on their resources, adding local resources, or isolating into subsystems). In case of such systems the shares of the participating organizations should depend both on their workload and on the owned resources; intuitively an organization that contributes many "useful" machines should be favored; similarly an organization that has only a few jobs.

Game theory is an established method for describing outcomes of decisions made by agents. If agents may form binding agreements, *cooperative game theory* studies the stability of resulting agreements (coalitions and revenues). There are well studied concepts of stability [30], like the *core*, the *kernel*, the *nucleolus*, the *stable set* or the *bargaining set*. The Shapley value [35] characterizes what is a *fair* distribution of the total revenue of the coalition between the participating agents.

The Shapley value has been used in scheduling theory but all the models we are aware of use the concept of money. The works of Carroll et at. [3], Mishra et al. [27], Mashayekhy and Grosu [26] and Moulin et al. [28] describe algorithms and the process of forming the coalitions for scheduling. These works assume that each job has a certain *monetary* value for the issuing organization and each organization has its initial monetary budget.

Money may have negative consequences on the stakeholders of resource-sharing consortia. Using (or even mentioning) money discourages people from cooperating [39]. This stays in sharp contrast with the idea behind the academic systems — sharing the infrastructure is a step towards closer cooperation. Additionaly, we believe that using money is inconvenient in non-academic systems as well. In many contexts, it is not clear how to valuate the completion of the job or the usage of a resource (especially when workload changes dynamically). We think that the accurate valuation is equally important (and perhaps equally difficult) as the initial problem of fair scheduling. Although auctions [2] or commodity markets [22] have been proposed to set prices, these approaches implicitly require to set the reference value to determine profitability. Other works on *monetary* game-theoretical models for scheduling include [8,9,12,13,31]; monetary approach is also used for other resource allocation problems, e.g. network bandwidth allocation [43]. However, none of these works describes how to valuate jobs and resources.

In a non-monetary approach proposed by Dutot el al. [6] the jobs are scheduled to minimize the global performance metric (the makespan) with an additional requirement — the utility of each player cannot be worse than if the player would act alone. Such approach ensures the stability of the system against actions of any single user (it is not profitable for the user to leave the system and to act alone) but not to the formation of sub-coalitions.

In the selfish job model [38] the agents are the jobs that selfishly choose processors on which to execute. Similarly to our model the resources are shared and treated as common good; however, no agent contributes resources.

An alternative to scheduling is to allow jobs to share resources concurrently. In congestion games [5,29,33] the utility of the player using a resource $R$ depends on the number of the players concurrently using $R$; the players are acting selfishly. Congestion games for divisible load scheduling were analyzed by Grosu and Chronopoulos [11] and Skowron and Rzadca [36].

In this paper we propose fair scheduling algorithms for systems composed of multiple organizations (in contrast to the case of multiple organizations using a system owned by a single entity). We model the organizations, their machines and their jobs as a cooperative game. In this game we do not use the concept of money. When measuring the contribution of the organization $O$ we analyze how the presence of $O$ in the grand coalition influences the completion times of the jobs of all participating organization. This contribution is expressed in the same units as the utility of the organization. In the design of the fair algorithm we use the concept of Shapley value. In contrast to simple cooperative game, in our case the value of the coalition (the total utility of the organizations in this coalition) depends on the underlying scheduling algorithm. This makes the problem of calculating the contributions of the organizations more involved. First we develop algorithms for arbitrary utilities (e.g. resource utilization, tardiness, flow time, etc.). Next we argue that designing the scheduling mechanism itself is not enough; we show that the utility function must be chosen to discourage organizations from manipulating their workloads (e.g. merging or spliting the jobs — similar ideas have been proposed for the money-based models [28]). We present an exponential scheduling algorithm for the strategy resilient utility function. We show that the fair scheduling problem is NP-hard and difficult to approximate. For a simpler case, when all the jobs are unit-size, we present a fully polynomial-time randomized approximation scheme (FPRAS). According to our experiments this algorithm is close to the optimum when used as a heuristics for workloads with different sizes of the jobs.

**Our contribution is the following:** (i) We derive the definition of the fair algorithm. Intuitively, fairness means that the utility for an organziation is close to its contribution i.e., its Shapley value. (ii) The Shapley value for an organization is derived based on the resources and the jobs the organization contributes — how the presence of the organization influences the processing times of the jobs of all organizations. The Shapley value is computed without any notion of money. (iii) We present an algorithm that computes a fair schedule for an arbitrary utility function (that might be a classical scheduling metric such as flow time, resource allocation, tardiness etc.). (iv) We observe that many utility functions are not strategy resistant, i.e. an organization can affect its utility by e.g. splitting its jobs into smaller pieces. We define the notion of strategy resistance and derive a strategy resistant utility function. (v) We show that the problem of calculating a fair schedule is NP-complete and hard to approximate. However, the problem parametrized by the number of organizations is fixed parameter tractable (FPT).

## 2. PRELIMINARIES

**Organizations, machines, jobs**. We consider a system built by a set of independent *organizations* $\mathcal{O} = \{O^{(1)}, O^{(2)}, \dots O^{(k)}\}$ [6]. Each organization $O^{(u)}$ owns a *computational cluster* consisting of $m^{(u)}$ *machines* (*processors*) denoted as $M_1^{(u)}, M_2^{(u)}, \dots M_{m^{(u)}}^{(u)}$ and produces its *jobs*, denoted as $J_1^{(u)}, J_2^{(u)}, \dots$. Each job $J_i^{(u)}$ has *release time* $r_i^{(u)} \in \mathbb{T}$, where $\mathbb{T}$ is a discrete set of time moments. We consider an on-line problem in which each job is unknown until its release time. We consider a non-clairvoyant model i.e., the job's processing time is unknown until the job completes (hence we do not need to use imprecise [24] run-time estimates). For the sake of simplicity of the presentation we assume that machines are identical, i.e. each job $J_i^{(u)}$ can be executed at any machine and its processing always takes $p_i^{(u)}$ time units; $p_i^{(u)}$ is the *processing time*. Most of the results, however, can be extended to the case of related machines, where $p_i^{(u)}$ is a function of the schedule – the only exception make the results in Section 5.1, where we rely on the

assumption that each job processed on any machine takes exactly one time unit. The results even generalize to the case of unrelated machines, however if we assume non-clairvoyant model with unrelated machines (i.e., we do not know the processing times of the jobs on any machine) then we cannot optimize the assignment of jobs to machines.

The jobs are sequential (this is a standard assumption in many scheduling models and, particularly, in the selfish job model [38]; an alternative is to consider the parallel jobs, which we plan to do in the future). Once a job is started, the scheduler cannot preempt it or migrate it to other machine (this assumption is usual in HPC scheduling because of high migration costs). Finally, we assume that the jobs of each individual organization should be started in the order in which they are presented. This allows organizations to have an internal prioritization of their jobs.

**Cooperation, schedules**. Organizations can cooperate and share their infrastructure; we say that organizations form a *coalition*. Formally, a coalition $\mathcal{C}$ is a subset of the set of all organizations, $\mathcal{C} \subseteq \mathcal{O}$. We also consider a specific coalition consisting of all organizations, which we call a *grand coalition* and denote as $\mathcal{C}_g$ (formally, $\mathcal{C}_g = \mathcal{O}$, but in some contexts we use the notation $\mathcal{C}_g$ to emphasize that we are referring to the set of the organizations that cooperate). A coalition must agree on the schedule of the jobs of its participants. A schedule $\sigma = \bigcup_{(u)\in\mathcal{C}} \bigcup_i \{(J_i^{(u)}, s_i^{(u)}, M(J_i^{(u)}))\}$ is a set of triples; a triple $(J_i^{(u)}, s_i^{(u)}, M(J_i^{(u)}))$ denotes a job $J_i^{(u)}$ started at time moment $s_i^{(u)} \geq r_i^{(u)}$ on machine $M(J_i^{(u)})$. Additionally, a machine executes at most one job at any time moment and each jobs is scheduled exactly once. We often identify a job $J_i^{(u)}$ with a pair $(s_i^{(u)}, p_i^{(u)})$; and a schedule with $\bigcup_{(u)} \bigcup_i \{(s_i^{(u)}, p_i^{(u)}))\}$ (we do so for a more compact presentation of our results). The coalition uses all the machines of its participants and schedules consecutive tasks on available machines. We consider only greedy schedules: at any time moment if there is a free machine and a non-empty set of ready, but not scheduled jobs, some job must be assigned to the free machine. Since we do not know neither the characteristics of the future workload nor the duration of the started but not yet completed jobs, any non-greedy policy would be suboptimal. Also, such greedy policies are used in real-world schedulers [19].

Let $\mathfrak{J}$ denote the set of all possible sets of the jobs. An *online scheduling algorithm* (in short a scheduling algorithm) $\mathcal{A} : \mathfrak{J} \times \mathbb{T} \to \mathcal{O}$ is an online algorithm that continuously builds a schedule: for a given time moment $t \in \mathbb{T}$ such that there is a free machine in $t$ and a set of jobs released before $t$ but not yet scheduled: $\mathcal{J} \in \mathfrak{J}$, $\mathcal{A}(\mathcal{J}, t)$ returns the organization the task of which should be started. The set of all possible schedules produced by such algorithms is the set of *feasible schedules* and it is denoted by $\Gamma$. We recall that in each feasible schedule the tasks of a single organization are started in a FIFO order.

**Objectives**. We consider a *utility function* $\psi : \Gamma \times \mathcal{O} \times \mathbb{T} \to \mathbb{R}$ that for a given schedule $\sigma \in \Gamma$, an organization $O^{(u)}$, and a time moment $t$ gives the value corresponding to the $O^{(u)}$ organization's satisfaction from a schedule $\sigma$ until $t$. The examples of such utility functions that are common in scheduling theory are: flow time, resource utilization, turnaround, etc. Our scheduling algorithms will only use the notions of the utilities and do not require any external payments.

Since a schedule $\sigma$ is fully determined by a scheduling algorithm $\mathcal{A}$ and a coalition $\mathcal{C}$, we often identify $\psi(\mathcal{A}, \mathcal{C}, O^{(u)}, t)$ with appropriate $\psi(\sigma, O^{(u)}, t)$. Also, we use a shorter notation $\psi^{(u)}(\mathcal{C})$ instead of $\psi(\mathcal{A}, \mathcal{C}, O^{(u)}, t)$ whenever the $\mathcal{A}$ and $t$ are known from the context. We define the *characteristic function* $v : \Gamma \times \mathbb{T} \to \mathbb{R}$ describing the total utility of the organizations from a schedule: $v(\mathcal{A}, \mathcal{C}, t) = \sum_{O^{(u)}\in\mathcal{C}} \psi(\mathcal{A}, \mathcal{C}, O^{(u)}, t)$. As above, we can use an equivalent formulation: $v(\sigma, t) = \sum_{O^{(u)}\in\mathcal{C}} \psi(\sigma, O^{(u)}, t)$, also using a shorter notations $v(\mathcal{C})$ whenever it is possible. Note that the utilities of the organizations $\psi^{(u)}(\mathcal{C})$ constitute a division of the value of the coalition $v(\mathcal{C})$.

# 3. FAIR SCHEDULING BASED ON THE SHAPLEY VALUE

In this section our goal is to find a scheduling algorithm $\mathcal{A}$ that at each time $t$ ensures a fair distribution of the coalition value $v(\mathcal{C})$ between the organizations. We will denote this desired fair division of the value $v$ as $\phi^{(1)}(v), \phi^{(2)}(v), \ldots, \phi^{(k)}(v)$ meaning that $\phi^{(u)}(v)$ denotes the ideally fair revenue (utility) obtained by organization $O^{(u)}$. The goal of the algorithm is to produce a schedule in which the actual utilities $\psi^{(1)}(\mathcal{C}), \psi^{(2)}(\mathcal{C}), \ldots \psi^{(k)}(\mathcal{C})$ are close to the desired fair division $\phi^{(1)}(v(\mathcal{C}), \phi^{(2)}(v(\mathcal{C}), \ldots, \phi^{(k)}(v(\mathcal{C})$ (we formalize this in Definitions 3.1 and 3.2).

We would like the values $\phi^{(u)}(v)$ to satisfy the following fairness properties (proposed by Shapley [35]):

1) efficiency: the total value $v(\mathcal{C})$ is distributed:

$$\sum_{O^{(u)}\in\mathcal{C}} \phi^{(u)}(v(\mathcal{C})) = v(\mathcal{C}).$$

2) symmetry: organizations $O^{(u)}$ and $O^{(u')}$ having indistinguishable contributions obtain the same profits:

$$\left(\forall_{\mathcal{C}'\subset\mathcal{C}: O^{(u)}, O^{(u')}\notin\mathcal{C}'}\ v(\mathcal{C}' \cup \{O^{(u)}\}) = v(\mathcal{C}' \cup \{O^{(u')}\})\right) \Rightarrow \left(\phi^{(u)}(v(\mathcal{C})) = \phi^{(u')}(v(\mathcal{C}))\right).$$

3) additivity: for any two characteristic functions $v$ and $w$ and a function $(v+w)$: $\forall_{\mathcal{C}'\subseteq\mathcal{C}}\ (v+w)(\mathcal{C}') = v(\mathcal{C}') + w(\mathcal{C}')$ we have that $\forall_{\mathcal{C}'\subseteq\mathcal{C}}\ \forall_u$: $\phi^{(u)}((v+w)(\mathcal{C})) = \phi^{(u)}(v(\mathcal{C})) + \phi^{(u)}(w(\mathcal{C}))$.
Consider any two independent schedules $\sigma_1$ and $\sigma_2$ that together form a schedule $\sigma_3 = \sigma_1 \cup \sigma_2$ ($\sigma_1$ and $\sigma_2$ are independent iff removing any subset of the jobs from $\sigma_1$ does not influence the completion time of any job in $\sigma_2$ and vice versa). The profit of an organization that participates only in one schedule (say $\sigma_1$) must be the same in case of $\sigma_1$ and $\sigma_3$ (intuitively: the jobs that do not influence the current schedule, also do not influence the current profits). The profit of every organization that participates in both schedules should in $\sigma_3$ be the sum of the profits in $\sigma_1$ and $\sigma_2$. Intuitively: if the schedules are independent then the profits are independent too.

4) dummy: an organization that does not increase the value of any coalition $C' \subset C$ gets nothing:

$$\left(\forall_{\mathcal{C}'\subset\mathcal{C}} : v(\mathcal{C}' \cup \{O^{(u)}\}) = v(\mathcal{C}')\right) \Rightarrow \phi^{(u)}(v(\mathcal{C})) = 0.$$

Since the four properties are actually the axioms of the Shapley value [35], they fully determine the single mapping between the coalition values and the profits of organizations (known as the Shapley value). In game theory the Shapley value is considered the classic mechanism ensuring the fair division of the revenue of the coalition. The Shapley value can be computed by the following

formula [35]:

$$\phi^{(u)}(v(\mathcal{C})) = \sum_{\mathcal{C}' \subseteq \mathcal{C} \setminus \{O^{(u)}\}} \frac{\|\mathcal{C}'\|!(\|\mathcal{C}\| - \|\mathcal{C}'\| - 1)!}{\|\mathcal{C}\|!} (v(\mathcal{C}' \cup \{O^{(u)}\}) - v(\mathcal{C}')) \quad (1)$$

**Algorithm 1:** Fair algorithm for arbitrary utility function $\psi$.

**Notation**:
**jobs**$[\mathcal{C}][O^{(u)}]$ — list of waiting jobs of organization $O^{(u)}$.
$\boldsymbol{\phi}[\mathcal{C}][O^{(u)}]$ — the contribution of $O^{(u)}$ in $\mathcal{C}$, $\phi^{(u)}(\mathcal{C})$.
$\boldsymbol{\psi}[\mathcal{C}][O^{(u)}]$ — utility of $O^{(u)}$ from being in $\mathcal{C}$, $\psi(\mathcal{C}, O^{(u)})$.
$\mathbf{v}[\mathcal{C}]$ — value of a coalition $\mathcal{C}$.
$\boldsymbol{\sigma}[\mathcal{C}]$ — schedule for a coalition $\mathcal{C}$.
`FreeMachine` $(\sigma, t)$ — returns true if and only if there is a free machine in $\sigma$ in time $t$.

```
ReleaseJob(O^(u), J):
    for C : O^(u) ∈ C do
        jobs[C][O^(u)].push(J)

Distance(C, O^(u), t):
    old ← σ[C];
    new ← σ[C] ∪ {(jobs[C][O^(u)].first, t)};
    Δψ ← ψ(new, O^(u), t) − ψ(old, O^(u), t);
    return |φ[C][O^(u)] + Δψ/||C|| − ψ[C][O^(u)] − Δψ|
    + Σ_{O^(u')} |φ[C][O^(u')] + Δψ/||C|| − ψ[C][O^(u')]|;

SelectAndSchedule(C, t):
    u ← argmin_{O^(u)}(Distance(C, O^(u), t)) ;
    σ[C] ← σ[C] ∪ {(jobs[C][u].first, t)};
    ψ[C][O^(u)] ← ψ(σ[C], O^(u), t);

UpdateVals(C, t):
    foreach O^(u) ∈ C do
        ψ[C][O^(u)] ← ψ(σ[C], O^(u), t);
        φ[C][O^(u)] ← 0;
    v[C] ← Σ_{O^(u)} ψ(σ[C], O^(u), t);
    foreach C_sub: C_sub ⊆ C do
        foreach O^(u) ∈ C_sub do
            φ[C][O^(u)] ← φ[C][O^(u)]+
                (v[C_sub] − v[C_sub \ {O^(u)}])
                .((||C_sub||−1)!(||C|−||C_sub||)!)/||C||! ;

FairAlgorithm(C):
    foreach time moment t do
        foreach job J_i^(u): r_i^(u) = t do
            ReleaseJob(O_i^(u), J_i^(u)) ;
        for s ← 1 to ||C|| do
            foreach C' ⊂ C, such that ||C'|| = s do
                UpdateVals(C', t);
                while FreeMachine(σ[C'], t) do
                    SelectAndSchedule(C', t);
                v[C] ← Σ_{O^(u)} ψ(σ[C], O^(u), t);
```

Let $\mathcal{L}_\mathcal{C}$ denote all orderings of the organizations from the coalition $\mathcal{C}$. Each ordering $\prec_\mathcal{C}$ can be associated with a permutation of the set $\mathcal{C}$, thus $\|\mathcal{L}_\mathcal{C}\| = \|\mathcal{C}\|!$. For the ordering $\prec_\mathcal{C} \in \mathcal{L}_C$ we define $\prec_\mathcal{C}(O^{(i)}) = \{O^{(j)} \in \mathcal{C} : O^{(j)} \prec_\mathcal{C} O^{(i)}\}$ as the set of all organizations from $\mathcal{C}$ that precede $O^{(i)}$ in the order $\prec_\mathcal{C}$. The Shapley value can be alternatively expressed [30] in the following form:

$$\phi^{(u)}(v(\mathcal{C})) = \frac{1}{\|\mathcal{C}\|!} \sum_{\prec_\mathcal{C} \in \mathcal{L}_\mathcal{C}} \left( v(\prec_\mathcal{C}(O^{(u)}) \cup \{O^{(u)}\}) - v(\prec_\mathcal{C}(O^{(u)})) \right). \quad (2)$$

This formulation has an interesting interpretation. Consider the organizations joining the coalition $\mathcal{C}$ in the order $\prec_\mathcal{C}$. Each organization $O^{(u)}$, when joining, contributes to the current coalition the value equal to $\left( v(\prec_\mathcal{C}(O^{(u)}) \cup \{O^{(u)}\}) - v(\prec_\mathcal{C}(O^{(u)})) \right)$. Intuitively, this value measures how the joining organization influences (decreases or increases) the total completion time of the jobs; $\phi^{(u)}(v(\mathcal{C}))$ is the expected contribution to the coalition $\mathcal{C}$, when the expectation is taken over the order in which the organizations join $\mathcal{C}$. Thus, we can identify the ideally fair utilities with the contributions of the organizations. Hereinafter we will call the value $\phi^{(u)}(v(\mathcal{C})$ (or using a shorter notation $\phi^{(u)}$) as the *contribution* of the organization $O^{(u)}$.

Informally speaking, we would like the utility of each organization $\psi$ to be as close to its contribution $\phi$ as possible. Ideally, the utilities of the organizations should be equal to the reference fair values, $\forall_u\ \psi^{(u)}(\mathcal{C}) = \phi^{(u)}(v(\mathcal{C}))$, but our scheduling problem is discrete so an algorithm guaranteeing this property may not exist. Thus, we will call as fair an algorithm that results in utilities close to contributions. We recall that the contribution is defined without a notion of money – the contribution of the organization measures how the presence of this organization affects the completion time of the jobs.

The following definition of a fair algorithm is in two ways recursive. First, we require an algorithm to be fair in all time moments $t$. Formally, a fair algorithm in time $t$ must also be fair in all previous time moments $t' < t$ (point 1.) [1]. Second, to assess the contribution of the organization to the coalition $\mathcal{C}$ (its Shapley value) we need to know how this organization, when joining, changes the schedule of each subcoalition $\mathcal{C}' \subset \mathcal{C}$. However, to determine the schedule for a subcoalition $\mathcal{C}'$ we need to know how a fair scheduling algorithm works for $\mathcal{C}'$. In other words, to define what is a fair algorithm for a coalition $\mathcal{C}$ we need to know what is a fair algorithm for all subcoalitions $\mathcal{C}' \subset \mathcal{C}$ (point 4.). Finally, assuming we know the fair algorithms for all subcoalitions and we have the contributions of the organizations calculated (point 3.), we look for an algorithm that minimizes the distance between the utilities and the contributions of the organizations (the argmin expression).

**Definition 3.1** *Set an arbitrary metric $\|\cdot\|_d : 2^k \times 2^k \to \mathbb{R}_{\geq 0}$; and set an arbitrary time moment $t \in \mathbb{T}$. $\mathcal{A}$ is a fair algorithm in $t$ for coalition $\mathcal{C}$ in metric $\|\cdot\|_d$ if and only if:*

$$\mathcal{A} \in \operatorname{argmin}_{\mathcal{A}' \in \mathcal{F}(<t)} \|\vec{\phi}(\mathcal{A}', \mathcal{C}, t) - \vec{\psi}(v(\mathcal{A}', \mathcal{C}, t)\|_d \quad \text{, where:}$$

1. *$\mathcal{F}(< t)$ is a set of algorithms fair in each point $t' < t$; $\mathcal{F}(< 0)$ is a set of all greedy algorithms,*
2. *$\vec{\psi}(v(\mathcal{A}', \mathcal{C})$ is a vector of utilities $\langle \psi^{(u)}(v(\mathcal{A}', \mathcal{C}))\rangle$,*
3. *$\vec{\phi}(\mathcal{A}', \mathcal{C})$ is a vector of contributions $\langle \phi^{(u)}(v(\mathcal{A}', \mathcal{C}))\rangle$, where $\phi^{(u)}(v(\mathcal{A}', \mathcal{C}))$ is given by Equation 1,*
4. *In Equation 1, for any $\mathcal{C}' \subset \mathcal{C}$, $v(\mathcal{C}')$ denotes $v(\mathcal{A}_f, \mathcal{C}')$, where $\mathcal{A}_f$ is any fair algorithm for $\mathcal{C}'$.*

[1] An alternative to being fair for all $t' < t$ would be to ensure asymptotic fairness; however, our formulation is more responsive and relevant for the online case. We want to avoid the case in which an organization is disfavored in one, possibly long, time period and favored in the next one.

**Definition 3.2** *$\mathcal{A}$ is a fair algorithm for coalition $\mathcal{C}$ if and only if it is fair in each time $t \in \mathbb{T}$.*

Further on, we consider algorithms fair in the Manhattan metric (our analysis can be generalized to other distance functions): $\|\vec{v_1}, \vec{v_2}\|_M = \sum_{i=1}^{k} |v_1[i] - v_2[i]|$.

Based on Definition 3.2 we construct a fair algorithm for an arbitrary utility function $\psi$ (Algorithm 1). The algorithm keeps a schedule for every subcoalition $\mathcal{C}' \subset \mathcal{C}$. For each time moment the algorithm complements the schedule starting from the subcoalitions of the smallest size. The values of all smaller coalitions $v[\mathcal{C}_s]$ are used to update the contributions of the organizations (lines 23-27) in the procedure `UpdateVals`). Before scheduling any job of the coalition $\mathcal{C}'$ the contribution and the utility of each organization in $\mathcal{C}'$ is updated (procedure `UpdateVals`). If there is a free machine and a set of jobs waiting for execution, the algorithm selects the job according to Definition 3.1, thus it selects the organization that minimizes the distance of the utilities $\vec{\psi}$ to their ideal values $\vec{\phi}$ (procedure `SelectAndSchedule`). Assuming the first job of the organization $O^{(u)}$ is tentatively scheduled, the procedure `Distance` computes a distance between the new values of $\vec{\psi}$ and $\vec{\phi}$. The procedure `Distance` works as follows. Assuming $O^{(u)}$ is selected the value $\Delta\psi$ denotes the increase of the utility of $O^{(u)}$ thanks to scheduling its first waiting job. This is also the increase of the value of the whole coalition. When procedure `Distance`$(\mathcal{C}, O^{(u)}, t)$ is executed, the schedules (and thus, the values) in time $t$ for all subcoalitions $\mathcal{C}' \subset \mathcal{C}$ are known. The schedule, for coalition $\mathcal{C}$ is known only in time $(t-1)$, as we have not yet decided which job should be scheduled in $t$. Thus, scheduling the job will change the schedule (and the value) only for a coalition $\mathcal{C}$. From Equation 1 it follows that if the value $v(\mathcal{C})$ of the coalition $\mathcal{C}$ increases by $\Delta\psi$ and the value of all subcoalitions remains the same, then the contribution $\phi^{(u')}$ of each organization $O^{(u')} \in \mathcal{C}$ to $\mathcal{C}$ will increase by the same value equal to $\Delta\psi/\|\mathcal{C}\|$. Thus, for each organization $O^{(u')} \in \mathcal{C}$ the new contribution of $O^{(u')}$ is $(\phi[\mathcal{C}][O^{(u')}] + \frac{\Delta\psi}{\|\mathcal{C}\|})$. The new utility for each organization $O^{(u')} \in \mathcal{C}$, such that $O^{(u')} \neq O^{(u)}$ is equal to $\psi[\mathcal{C}][O^{(u')}]$. The new utility of the organization $O^{(u)}$ is equal to $(\psi[\mathcal{C}][O^{(u)}] + \Delta\psi)$.

**Theorem 3.3** *Algorithm 1 is a fair algorithm.*

PROOF. Algorithm 1 is a straightforward implementation of Definition 3.2. □

**Proposition 3.4** *In each time moment $t$ the time complexity of Algorithm 1 is $O(\|\mathcal{O}\|(2^{\|\mathcal{O}\|}\sum m^{(u)} + 3^{\|\mathcal{O}\|}))$.*

PROOF. Once the contribution is calculated, each coalition in $t$ may schedule at most $\sum m^{(u)}$ jobs. The time needed for selecting each such a job is proportional to the number of the organizations. Thus, we get the $\|\mathcal{O}\|2^{\|\mathcal{O}\|}\sum m^{(u)}$ part of the complexity. For calculating the contribution of the organization $O^{(u)}$ to the coalition $\mathcal{C}$ the algorithm considers all subsets of $\mathcal{C}$ – there are $2^{\|\mathcal{C}\|}$ such subsets. Since there are $\binom{\|\mathcal{O}\|}{k}$ coalitions of size $k$, the number of the operations required for calculating the contributions of all organizations is proportional to:

$$\sum_{(u)}\sum_{k=0}^{\|\mathcal{O}\|}\binom{\|\mathcal{O}\|}{k}2^k = \|\mathcal{O}\|\sum_{k=0}^{\|\mathcal{O}\|}\binom{\|\mathcal{O}\|}{k}1^{\|\mathcal{O}\|-k}2^k =$$
$$\|\mathcal{O}\|(1+2)^{\|\mathcal{O}\|} = \|\mathcal{O}\|3^{\|\mathcal{O}\|}.$$

This gives the $\|\mathcal{O}\|3^{\|\mathcal{O}\|}$ part of the complexity and completes the proof. □

**Corollary 3.5** *The problem of finding fair schedule parametrized with the number of organizations is FPT.*

# 4. STRATEGY-PROOF UTILITY FUNCTIONS

There are many utility functions considered in scheduling, e.g. flow time, turnaround time, resource utilization, makespan, tardiness. However, it is not sufficient to design a fair algorithm for an arbitrary utility function $\psi$. Some functions may create incentive for organizations to manipulate their workload: to divide the tasks into smaller pieces, to merge or to delay them. This is undesired as an organization should not profit nor suffer from the way it presents its workload. An organization should present their jobs in the most convenient way; and should not play against other organizations. We show that in multi-organizational systems, as we have to take into account such manipulations, the choice of the utility functions is restricted.

For the sake of this section we introduce additional notation: let us fix an organzation $O^{(u)}$ and let $\sigma_t$ denote a schedule of the jobs of $O^{(u)}$ in time $t$. The jobs $J_i(s_i, p_i)$ of $O^{(u)}$ are characterized by their start times $s_i$ and processing times $p_i$. We are considering envy-free utility functions that for a given organization $O^{(u)}$ depend only on the schedule of the jobs of $O^{(u)}$. This means that there is no external economical relation between the organization (the organization $O^u$ cares about $O^v$ only if the jobs of $O^v$ influence the jobs of $O^u$ – in contrast to looking directly at the utility of $O^v$). We also assume the non-clairvoyant model – the utility in time $t$ depends only on the jobs or the parts of the jobs completed before or at $t$. Let us assume that our goal is to maximize the utility function[2]. We start from presenting the desired properties of the utility function $\psi$ (when presenting the properties we use the shorter notation $\psi(\sigma_t)$ for $\psi(\sigma_t, t)$):

1) Tasks anonymity (starting times) — improving the completion time of a single task with a certain processing time $p$ by one unit of time is for each task equally profitable – for $s, s' \leq t-1$, we require:

$$\psi(\sigma_t \cup \{(s,p)\}) - \psi(\sigma_t \cup \{(s+1,p)\}) =$$
$$\psi(\sigma'_t \cup \{(s',p)\}) - \psi(\sigma'_t \cup \{(s'+1,p)\}) > 0.$$

2) Tasks anonymity (number of tasks) — in each schedule increasing the number of completed tasks is equally profitable – for $s \leq t-1$, we require:

$$\psi(\sigma_t \cup \{(s,p)\}) - \psi(\sigma_t) = \psi(\sigma'_t \cup \{(s,p)\}) - \psi(\sigma'_t) > 0.$$

3) Strategy-resistance — the organization cannot profit from merging multiple smaller jobs into one larger job or from dividing a larger job into smaller pieces:

$$\psi(\sigma_t \cup \{(s,p_1)\}) + \psi(\sigma_t \cup \{(s+p_1,p_2)\}) =$$
$$\psi(\sigma_t \cup \{(s,p_1+p_2)\}).$$

In spite of dividing and merging the jobs, each organization can delay the release time of their jobs and artificially increase the size of the jobs. Delaying the jobs is however never profitable for the organization (by property 1). Also, the strategy-resistance property discourages the organizations to increase the sizes of their jobs (the utility coming from processing a larger job is always greater).

[2]We can transform the problem to the minimization form by taking the inverse of the standard maximization utility function

**Algorithm 2:** Function `SelectAndSchedule` for utility function $\psi_{sp}$.

```
1 SelectAndSchedule(C, t):
2     u ← argmin_{O^(u)}(ψ[C][O^(u)] − φ[C][O^(u)]) ;
3     σ[C] ← σ[C] ∪ {(jobs[C][u].first, t)};
4     ψ[C][O^(u)] ← ψ(σ[C], O^(u), t);
```

To within a multiplicative and additive constants, there is only one utility function satisfying the aforementioned properties.

**Theorem 4.1** *Let $\psi$ be a utility function that satisfies the 3 properties: task anonymity (starting times); task anonymity (number of tasks); strategy-resistance. $\psi$ is of the following form:*

$$\psi(\sigma,t)=\sum_{(s,p)\in\sigma_t}\min(p,t-s)(K_1-K_2\frac{s+\min(s+p-1,t-1)}{2})+K_3$$

*where*

1. $K_1=\psi(\sigma\cup\{(0,1)\},t)-\psi(\sigma)>0$,
2. $K_2=\psi(\sigma\cup\{(s,p)\},t)-\psi(\sigma\cup\{(s+1,p)\},t)>0$,
3. $K_3=\psi(\emptyset)$.

PROOF. Proof is in the full version of this paper [37]. □

We set the constants $K_1, K_2, K_3$ so that to simplify the form of the utility function and ensure that the utility is always positive. With $K_1=1$, $K_2=t$ and $K_3=0$, we get the following strategy-proof utility function:

$$\psi_{sp}(\sigma,t)=\sum_{(s,p)\in\sigma:s\le t}\min(p,t-s)\left(t-\frac{s+\min(s+p-1,t-1)}{2}\right) \tag{3}$$

$\psi_{sp}$ can be interpreted as the task throughput. A task with processing time $p_i$ can be identified with $p_i$ unit-sized tasks starting in consecutive time moments. Intuitively, the function $\psi_{sp}$ assigns to each such unit-sized task starting at time $t_s$ a utility value equal to $(t-t_s)$; the higher the utility value, the earlier this unit-sized task completes. A utility of the schedule is the sum of the utilities over all such unit-sized tasks. $\psi_{sp}$ is similar to the flow time except for two differences: (i) Flow time is a minimization objective, but increasing the number of completed jobs increases its value. E.g., scheduling no jobs results in zero (optimal) flow time, but of course an empty schedule cannot be considered optimal (breaking the second axiom); (ii) Flow time favors short tasks, which is an incentive for dividing tasks into smaller pieces (this breaks strategy-resistance axiom). The differences between the flow time and $\psi_{sp}$ is also presented on example in Figure 1. The similarity of $\psi_{sp}$ to the flow time is quantified by Proposition 4.2 below.

**Proposition 4.2** *Let $\mathcal{J}$ be a fixed set of jobs, each having the same processing time $p$ and each completed before $t$. Then, maximization of the $\psi_{sp}$ utility is equivalent to minimization the flow time of the jobs.*

PROOF. Proof is in the full version of this paper [37]. □

# 5. FAIR SCHEDULING WITH STRATEGY-PROOF UTILITY

For the concrete utility $\psi_{sp}$ we can simplify the `SelectAndSchedule` function in Algorithm 1. The simplified version is presented in Algorithm 2.

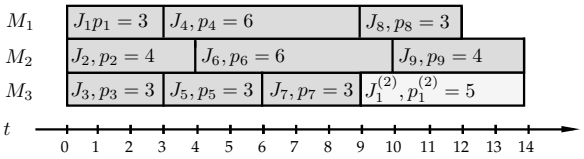


**Figure 1: Consider 9 jobs owned by $O^{(1)}$ and a single job owned by $O^{(2)}$, all scheduled on 3 processors. We assume all jobs were released in time 0. In this example all jobs finish before or at time $t=14$. The utility $\psi_{sp}$ of the organization $O^{(1)}$ in time 13 does not take into account the last uncompleted unit of the job $J_9$, thus it is equal to: $3\cdot(13-\frac{0+2}{2})+4\cdot(13-\frac{0+3}{2})+\cdots+3\cdot(13-\frac{9+11}{2})+3\cdot(13-\frac{10+12}{2})=262$. The utility in time 14 takes into account all the parts of the jobs, thus it is equal to $3\cdot(14-\frac{0+2}{2})+4\cdot(14-\frac{0+3}{2})+\cdots+3\cdot(14-\frac{9+11}{2})+4\cdot(14-\frac{10+13}{2})=297$. The flow time in time 14 is equal to $3+4+\cdots+14=70$. If there was no job $J_1^{(2)}$, then $J_9$ would be started in time 9 instead of 10 and the utility $\psi_{sp}$ in time 14 would increase by $4\cdot(\frac{10+13}{2}-\frac{9+12}{2})=4$ (the flow time would decrease by 1). If, for instance, $J_6$ was started one time unit later, then the utility of the schedule would decrease by 6 (the flow time would decrease by 1), which shows that the utility takes into account the sizes of the jobs (in contrast to the flow time). If the job $J_9$ was not scheduled at all, the utility $\psi_{sp}$ would decrease by 10, which shows that the schedule with more tasks has higher (more optimal) utility (the flow time would decrease by 14; since flow time is a minimization metric, this breaks the second axiom regarding the tasks anonymity).**

The algorithm selects the organization $O^{(u)}$ that has the largest difference $(\phi^{(u)}-\psi^{(u)})$ that is the organization that has the largest contribution in comparison to the obtained utility. One can wonder whether we can select the organization in polynomial time – without keeping the $2^{\|\mathcal{C}\|}$ schedules for all subcoalitions. Unfortunately, the problem of calculating the credits for a given organization is NP-hard.

**Theorem 5.1** *The problem of computing the contribution $\phi^{(u)}(\mathcal{C},t)$ for a given organization $O^{(u)}$ in coalition $\mathcal{C}$ in time $t$ is NP-hard.*

PROOF. We present the reduction of the SUBSETSUM problem (which is NP-hard) to the problem of calculating the contribution for an organization. Let $I$ be an instance of the SUBSETSUM problem. In $I$ we are given a set of $k$ integers $S=\{x_1,x_2,\ldots,x_k\}$ and a value $x$. We ask whether there exists a subset of $S$ with the sum of elements equal to $x$. From $I$ we construct an instance $I_{con}$ of the problem of calculating the contribution for a given organization. Intuitively, we construct the set of $(\|S\|+2)$ organizations: $\|S\|$ of them will correspond to the appropriate elements from $S$. The two dummy organizations $a$ and $b$ are used for our reduction. One dummy organization $a$ has no jobs. The second dummy organization $b$ has a large job that dominates the value of the whole schedule. The instance $I_{con}$ is constructed in such a way that for each coalition $\mathcal{C}$ such that $b\in\mathcal{C}$ and such that the elements of $S$ corresponding to the organizations from $\mathcal{C}$ sum up to the value lower than $x$, the marginal contribution of $a$ to $\mathcal{C}$ is $L+O(L)$, where $O(L)$ is small in comparison with $L$. The marginal contribution of $a$ to other coalitions is small $(O(L))$. Thus, from the contribution of $a$, we can count the subsets of $S$ with the sum of the elements lower than $x$. By repeating this procedure for $(x+1)$ we can count the subsets of $S$ with the sum of the elements lower than $(x+1)$. By comparing the two values, we can find whether

there exists the subset of $S$ with the sum of the elements equal to $x$. The precise construction is described below.

Let $\mathcal{S}_{<x} = \{S' \subset S : \sum_{x_i \in S'} s_i < x\}$ be the set of the subsets of $S$, each having the sum of the elements lower than $x$. Let $n_{<x}(S) = \sum_{S' \in \mathcal{S}_{<x}} (\|S'\|+1)!(\|S\| - \|S'\|)!$ be the number of the orderings (permutations) of the set $S \cup \{a, b\}$ that starts with some permutation of the sum of exactly one element of $\mathcal{S}_{<x}$ (which is some subset of $S$ such that the sum of the elements of this subset is lower than $x$) and $\{b\}$ followed by the element $a$. In other words, if we associate the elements from $S \cup \{a, b\}$ with the organizations and each ordering of the elements of $S \cup \{a, b\}$ with the order of the organizations joining the grand coalition, then $n_{<x}(S)$ is the number of the orderings corresponding to the cases when organization $a$ joins grand coalition just after all the organizations from $S' \cup \{b\}$, where $S'$ is some element of $\mathcal{S}_{<x}$. Of course $\mathcal{S}_{<x} \subseteq \mathcal{S}_{<(x+1)}$. Note that there exists $S' \subset S$, such that $\sum_{x_i \in S'} x_i = x$ if and only if the set $\mathcal{S}_{<x}$ is a proper subset of $\mathcal{S}_{<(x+1)}$ (i.e. $\mathcal{S}_{<x} \subsetneq \mathcal{S}_{<(x+1)}$). Indeed, there exists $S'$ such that $S' \notin \mathcal{S}_{<x}$ and $S' \in \mathcal{S}_{<(x+1)}$ if and only if $\sum_{x_i \in S'} x_i < x+1$ and $\sum_{x_i \in S'} x_i \geq x$ from which it follows that $\sum_{x_i \in S'} x_i = x$. Also, $\mathcal{S}_{<x} \subsetneq \mathcal{S}_{<(x+1)}$ if and only if $n_{<(x+1)}(S)$ is greater than $n_{<(x)}(S)$ (we are doing a summation of the positive values over the larger set).

In $I_{con}$ there is a set of $(k+2)$ machines, each owned by a different organization. We will denote the set of first $k$ organizations as $\mathcal{O}_S$, the (k+1)-th organization as $a$ and the (k+2)-th organization as $b$. Let $x_{tot} = \sum_{j=1}^{k} x_j + 2$. The $i$-th organization from $\mathcal{O}_S$ has 4 jobs: $J_1^{(i)}, J_2^{(i)}, J_3^{(i)}$ and $J_4^{(i)}$, with release times $r_1^{(i)} = r_1^{(i)} = 0$, $r_3^{(i)} = 3$ and $r_4^{(i)} = 4$; and processing times $p_1^{(i)} = p_2^{(i)} = 1$, $p_3^{(i)} = 2x_{tot}$ and $p_4^{(i)} = 2x_i$. The organization $a$ has no jobs; the organization $b$ has two jobs $J_1^{(b)}$ and $J_2^{(b)}$, with release times $r_1^{(b)} = 2$ and $r_2^{(b)} = (2x+3)$; and processing times $p_1^{(b)} = (2x+2)$ and $p_2^{(b)} = L = 4\|S\|x_{tot}^2((k+2)!) + 1$ (intuitively $L$ is a large number).

Until time $t = 2$ only the organizations from $\mathcal{O}_S$ have some (unit-size) jobs to be executed. The organization $b$ has no jobs till time $t = 2$, so it will run one or two unit-size jobs of the other organizations, contributing to all such coalitions that include $b$ and some other organizations from $\mathcal{O}_S$. This construction allows to enforce that in the first time moment after $t = 2$ when there are jobs of some of the organizations from $\mathcal{O}_S$ and of $b$ available for execution, the job of $b$ will be selected and scheduled first.

Let us consider a contribution of $a$ to the coalition $\mathcal{C}$ such that $a \notin \mathcal{C}$ and $b \in \mathcal{C}$. There are $(\|\mathcal{C} \cap \mathcal{O}_S\| + 2)$ machines in the coalition $\mathcal{C} \cup \{a\}$. The schedule in $\mathcal{C} \cup \{a\}$ after $t = 2$ looks in the following way (this schedule is depicted in Figure 2). In time $t = 2$ one machine (let us denote this machine as $M'$) starts the job $J_1^{(b)}$ In time $t = 3$ some $\|\mathcal{C} \cap \mathcal{O}_S\|$ machines start the third jobs (the one with size $2x_{tot}$) of the organizations from $\mathcal{C} \cap \mathcal{O}$ and one machine (denoted as $M''$) starts the fourth jobs of the organizations from $\mathcal{C} \cap \mathcal{O}_S$; the machine $M''$ completes processing all these jobs in time $2y+4$, where $y = \sum_{i:O^{(i)} \in \mathcal{C} \wedge O^{(i)} \in \mathcal{O}_S} x_i$ (of course $2y+4 \leq 2x_{tot}$). In time $(2x+3)$, if $y < x$ the machine $M''$ starts processing the large job $J_2^{(b)}$ of the organization $b$; otherwise machine $M''$ in time $(2x+3)$ still executes some job $J_4^{(i)}$ (as the jobs $J_4^{(i)}$ processed on $M''$ start in even time moments). In time $2x+4$, if $y \geq x$, the large job $J_2^{(b)}$ is started by machine $M'$ just after the job $J_1^{(b)}$ is completed, ($J_1^{(b)}$ completes in $(2x+4)$); here we use the fact that after $t = 2$, $b$ will be prioritized over the organizations from $\mathcal{O}_S$. To sum up: if $y < x$ then the large job $J_2^{(b)}$ is started in time $(2x+3)$, otherwise it is started in time $(2x+4)$.

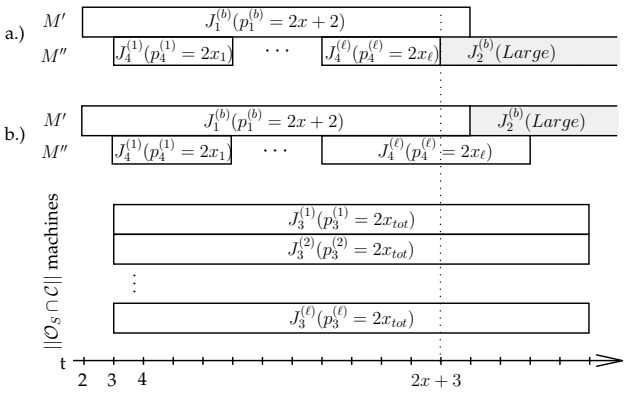


**Figure 2: The schedules for the coalition $\mathcal{C} \cup \{a\}$ for two cases: a) $\sum_{i:O^{(i)} \in \mathcal{C} \wedge O^{(i)} \in \mathcal{O}_S} x_i \leq x$, b) $\sum_{i:O^{(i)} \in \mathcal{C} \wedge O^{(i)} \in \mathcal{O}_S} x_i > x$. The two cases a) and b) differ only in the schedules on machines $M'$ and $M''$. In the case a) the large job $J_2^{(b)}$ (marked as a light gray) is started one time unit earlier than in case b).**

If $y < x$ then by considering only a decrease of the starting time of the largest job, the contribution of $a$ to the coalition $\mathcal{C}$ can be lower bounded by $c_1$:

$$c_1 = L\left(t - \frac{(2x+3) + (2x+3+L)}{2}\right) - L\left(t - \frac{(2x+4) + (2x+4+L))}{2}\right) = L,$$

The organization $a$ causes also a decrease of the starting times of the small jobs (the jobs of the organizations from $\mathcal{O}_S$); each job of size smaller or equal to $2x_{tot}$. The starting time of each such small job is decreased by at most $2x_{tot}$ time units. Thus, the contribution of $a$ in case $y < x$ can be upper bounded by $c_2$:

$$c_2 \leq L + 4\|S\|x_{tot}^2.$$

If $y \geq x$ then $a$ causes only a decrease of the starting times of the small jobs of the organizations from $\mathcal{O}_S$, so the contribution of $a$ to $\mathcal{C}$ in this case can be upper bounded by $c_3$:

$$c_3 \leq 4\|S\|x_{tot}^2.$$

By similar reasoning we can see that the contribution of $a$ to any coalition $\mathcal{C}'$ such that $b \notin \mathcal{C}'$ is also upper bounded by $4\|S\|x_{tot}^2$.

The contribution of organization $a$, $\phi^{(a)}$, is given by Equation 1, with $u = a$ and $C = \{O^{(1)} \ldots O^{(k+2)}\}$. Thus:

$$\phi^{(a)} = \sum_{\mathcal{C}' \subseteq \mathcal{C} \setminus \{a\}} \frac{\|\mathcal{C}'\|!(k+1-\|\mathcal{C}'\|)!}{(k+2)!} \text{marg\_}\phi(\mathcal{C}', a),$$

where $\text{marg\_}\phi(\mathcal{C}', a)$ is the contribution of $a$ to coalition $\mathcal{C}'$. All the coalitions $\mathcal{C}'$ such that $a \notin \mathcal{C}', b \in \mathcal{C}'$ and $\sum_{i:O^{(i)} \in \mathcal{C}' \cap \mathcal{O}_S} x_i < x$ will contribute to $\phi^{(a)}$ the value at least equal to $\frac{n_{<x}(S)}{(k+2)!}c_1 = \frac{n_{<x}(S)L}{2(k+2)!}$ (as there is exactly $n_{<x}(S)$ orderings corresponding to the case when $a$ is joining such coalitions $\mathcal{C}'$) and at most equal to $\frac{n_{<x}(S)}{(k+2)!}c_2 \leq \frac{n_{<x}(S)(L+8\|S\|x_{tot}^2)}{2(k+2)!}$. The other $(k+2)! - n_{<x}(S)$ orderings will contribute to $\phi^{(a)}$ the value at most equal to $\frac{((k+2)!-n_{<x}(S))}{(k+2)!}c_3 = \frac{((k+2)!-n_{<x}(S))(4\|S\|x_{tot}^2)}{(k+2)!}$. Also:

$$\frac{((k+2)! - n_{<x}(S))(4\|S\|x_{tot}^2)}{(k+2)!} + \frac{n_{<x}(S)(4\|S\|x_{tot}^2)}{(k+2)!} =$$

$$4\|S\|x_{tot}^2 < \frac{L}{(k+2)!},$$

which means that $\phi^{(a)}$ can be stated as $\phi^{(a)} = \frac{n_{<x}(S)L}{(k+2)!} + R$, where $0 \leq R \leq \frac{L}{(k+2)!}$. We conclude that $\lfloor \frac{(k+2)!\phi^{(a)}}{L} \rfloor = n_{<x}(S)$. We have shown that calculating the value of $\phi^{(a)}$ allows us to find the value $n_{<x}(S)$. Analogously, we can find $n_{<(x+1)}(S)$. By comparing $n_{<x}(S)$ with $n_{<(x+1)}(S)$ we find the answer to the initial SUBSETSUM problem, which completes the proof.

□

We propose the following definition of the approximation of the fair schedule (similar definitions of the approximation ratio are used for multi-criteria optimization problems [7]):

**Definition 5.2** *Let $\sigma$ be a schedule and let $\vec{\psi}$ be a vector of the utilities of the organizations in $\sigma$. We say that $\sigma$ is an $\alpha$-approximation fair schedule in time $t$ if and only if there exists a truly fair schedule $\sigma^*$, with the vector $\vec{\psi}^* = \langle \psi^{(u),*} \rangle$ of the utilities of the organizations, such that:*

$$\|\vec{\psi} - \vec{\psi}^*\|_M \leq \alpha \|\vec{\psi}^*\|_M = \alpha \sum_u \psi^{(u),*} = \alpha \cdot v(\sigma^*, \mathcal{C}).$$

Unfortunately, the problem of finding the fair schedule is difficult to approximate. There is no algorithm better than 1/2 (the proof below). This means that the problem is practically inapproximable. Consider two schedules of jobs of $m$ organizations on a single machine. Each organization has one job; all the jobs are identical. In the first schedule $\sigma_{ord}$ the jobs are scheduled in order: $J_1^{(1)}, J_1^{(2)}, \ldots J_1^{(m)}$ and in the second schedule $\sigma_{rev}$ the jobs are scheduled in exactly reverse order: $J_1^{(m)}, J_1^{(m-1)}, \ldots J_1^{(1)}$. The relative distance between $\sigma_{ord}$ and $\sigma_{rev}$ tends to 1 (with increasing $m$), so $(\frac{1}{2})$-approximation algorithm does not allow to decide whether $\sigma_{ord}$ is truly better than $\sigma_{rev}$. In other words, $(\frac{1}{2})$-approximation algorithm cannot distinguish whether a given order of the priorities of the organizations is more fair then the reverse order.

**Theorem 5.3** *For every $\epsilon > 0$, there is no polynomial algorithm for finding the $(\frac{1}{2} - \epsilon)$-approximation fair schedule, unless* P = NP.

PROOF SKETCH. Intuitively, we divide time in a number of independent batches. The jobs in the last batch are significantly larger than all the previous ones. We construct the jobs in all first batches so that the order of execution of the jobs in the last batch depends on whether there exists a subset $S' \subset S$ such that $\sum_{x_i \in S'} x_i = x$. If the subset does not exist the organizations are prioritized in some predefined order $\sigma_{ord}$; otherwise, the order is reversed $\sigma_{rev}$. The sizes of the jobs in the last batch are so large that they dominate the values of the utilities of the organizations. The relative distance between the utilities in $\sigma_{ord}$ and in $\sigma_{rev}$ is $(1 - \epsilon)$ so any $(\frac{1}{2} - \epsilon)$-approximation algorithm $\mathcal{A}$ would allow to infer the true fair schedule for such constructed instance, and so the answer to the initial SUBSETSUM problem. The precise construction is described the full version of this paper [37]. □

## 5.1 Special case: unit-size jobs

In case when the jobs are unit-size the problem has additional properties that allow us to construct an efficient approximation (however, the complexity of this special case is open). However, the results in this section do not generalize to related or unrelated processors. For unit-size jobs, the value of each coalition $v(\mathcal{C})$ does not depend on the schedule:

**Proposition 5.4** *For any two greedy algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, for each coalition $\mathcal{C}$ and each time moment $t$, the values of the coalitions $v(\mathcal{A}_1, \mathcal{C}, t)$ and $v(\mathcal{A}_2, \mathcal{C}, t)$ are equal, provided all jobs are unit-size.*

PROOF. We prove the following stronger thesis: for every time moment $t$ any two greedy algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ schedule the same number of the jobs till $t$. We prove this thesis by induction. The base step for $t = 0$ is trivial. Having the thesis proven for $(t-1)$ and, thus knowing that in $t$ in both schedules there is the same number of the jobs waiting for execution (here we use the fact that the jobs are unit-size), we infer that in $t$ the two algorithms schedule the same number of the jobs. Since the value of the coalition does not take into account the owner of the job, we get the thesis for $t$. This completes the proof. □

As the result, we can use the randomized approximation algorithm for the scheduling problem restricted to unit-size jobs (Algorithm 3). The algorithm is inspired by the randomized approximation algorithm for computing the Shapley value presented by Liben-Nowell et al [25]. However, in our case, the game is not supermodular (which is shown in Proposition 5.5 below), and so we have to adapt the algorithm and thus obtain different approximation bounds.

**Proposition 5.5** *In case of unit-size jobs the cooperation game in which the value of the coalition $\mathcal{C}$ is $v(\mathcal{C}) = \sum_{O^{(u)} \in \mathcal{C}} \psi(O^{(u)})$ is not supermodular.*

PROOF. Proof is in the full version of this paper [37]. □

In this algorithm we keep simplified schedules for a random subset of all possible coalitions. For each organization $O^{(u)}$ the set $Subs[O^{(u)}]$ keeps $N = \frac{\|\mathcal{C}\|^2}{\epsilon^2} \ln\left(\frac{\|\mathcal{C}\|}{1-\lambda}\right)$ random coalitions not containing $O^{(u)}$; for each such random coalition $\mathcal{C}'$ which is kept in $Subs[O^{(u)}]$, $Subs'[O^{(u)}]$ contains the coalition $\mathcal{C}' \cup \{O^{(u)}\}$. For the coalitions kept in $Subs[O^{(u)}]$ we store a simplified schedule (the schedule that is determined by an arbitrary greedy algorithm). The simplified schedule allows us to find the value $v(\mathcal{C}')$ of the coalition $\mathcal{C}'$. Maintaining the whole schedule would require the recursive information about the schedules in the subcoalitions of $\mathcal{C}'$. However, as the consequence of Proposition 5.4 we know that the value of the coalition $v(\mathcal{C}')$ can be determined by an arbitrary greedy algorithm[3].

The third $foreach$ loop in procedure `FairAlgorithm` (line 26 in Algorithm 3) updates the values of all coalitions kept in $Subs$ and $Subs$'. From Equation 3 it follows that after one time unit if no additional job is scheduled, the value of the coalition increases by the number of completed unit-size parts of the jobs (here, as the jobs are unit size, $\mathrm{finPerCoal}[\mathcal{C}']$ is the number of the completed jobs). In time moment $t$, all waiting jobs (the number of such jobs is $\|\mathrm{jobs}[\mathcal{C}][O^{(u)}]\|$) are scheduled provided there are enough processors (the number of the processors is $\sum_{O^{(u)} \in \mathcal{C}'} m^{(u)}$). If $n$ additional jobs are scheduled in time $t$ then the value of the coalition in time $t$ increases by $n$.

In the fourth $foreach$ loop (line 32 in Algorithm 3), once again we use the fact that the utility of the organization after one time unit increases by the number of finished jobs ($\mathrm{finPerOrg}[O^{(u)}]$). In the last $foreach$ loop (line 35) the contribution of the organization

[3]In this point we use the assumption about the unit size of the jobs. The algorithm cannot be extended to the general case. In a general case, for calculating the value for each subcoalition we would require the exact schedule which cannot be determined polynomially (Theorem 5.1).

**Algorithm 3:** Fair algorithm for arbitrary utility function for utility function $\psi_{sp}$ and for unit-size jobs.

**Notation**:
$\epsilon$, $\lambda$ — as in Theorem 5.6

```
Prepare(C):
    N ← ⌈(‖C‖²/ε²) ln(‖C‖/(1−λ))⌉;
    Γ ← generate N random orderings (permutations) of the
      set of all organizations (with replacement);
    Subs ← Subs' ← ∅ ;
    foreach ≺ ∈ Γ do
        for u ← 1 to ‖C‖ do
            C' ← {O^(i) : O^(i) ≺ O^(u)} ;
            Subs ← Subs ∪ {C'};
            Subs' ← Subs' ∪ {C' ∪ {O^(u)}} ;

ReleaseJob(O^(u), J):
    for C' ∈ Subs ∪ Subs' : O^(u) ∈ C' do
        jobs[C'][O^(u)].push(J)

SelectAndSchedule(C, t):
    u ← argmin_{O^(u)}(ψ[C][O^(u)] − φ[C][O^(u)]) ;
    σ[C] ← σ[C] ∪ {(jobs[C][u].first, t)};
    finPerOrg[O^(u)] ← finPerOrg[O^(u)] + 1;
    φ[O^(u)] ← φ[O^(u)] + 1;

FairAlgorithm(C):
    Prepare(C) ;
    foreach time moment t do
        foreach job J_i^(u): r_i^(u) = t do
            ReleaseJob(O_i^(u), J_i^(u));
        foreach C' ⊂ Subs ∪ Subs' do
            v[C'] ← v[C'] + finPerCoal[C'] ;
            n ← min(Σ_{O^(u)∈C'} m^(u), ‖jobs[C][O^(u)]‖) ;
            remove first n jobs from jobs[C][O^(u)] ;
            finPerCoal[C'] ← finPerCoal[C'] + n ;
            v[C'] ← v[C'] + n ;
        foreach O^(u) ∈ C do
            ψ[O^(u)] ← ψ[O^(u)] + finPerOrg[O^(u)];
            φ[O^(u)] ← 0;
            foreach C' ∈ Subs : O^(u) ∉ C' do
                marg_φ ← v[C' ∪ {O^(u)}] − v[C'] ;
                φ[O^(u)] ← φ[O^(u)] + marg_φ · 1/N;
        while FreeMachine(σ[C], t) do
            SelectAndSchedule(C,t);
```

is approximated by summing the marginal contributions $\mathrm{marg}\_\phi$ only for the kept coalitions. Theorem 5.6 below gives the bounds for the quality of approximation.

**Theorem 5.6** *Let $\vec{\psi}$ denote the vector of utilities in the schedule determined by Algorithm 3. If the jobs are unit-size, then $\mathcal{A}$ with the probability $\lambda$ determines the $\epsilon$-approximation schedule, i.e. gives guarantees for the bound on the distance to the truly fair solution: $\|\vec{\psi} - \vec{\psi}^*\|_M \leq \epsilon|\vec{\psi}^*|$.*

PROOF. Proof is in the full version of this paper [37]. □

The complexity of Algorithm 3 is $\|\mathcal{O}\| \cdot N = \|\mathcal{O}\| \frac{\|\mathcal{C}\|^2}{\epsilon^2} \ln\left(\frac{\|\mathcal{C}\|}{1-\lambda}\right)$ times the complexity of the single-organization scheduling algorithm. As a consequence, we get the following result:

**Corollary 5.7** *There exists an FPRAS for the problem of finding the fair schedule for the case when the jobs are unit size.*

In the full version of this paper [37] we show that Algorithm 3 can be used as a heuristic for the general case and that it produces more fair schedules than the round robin algorithm.

# 6. CONCLUSIONS

In this paper we define the fairness of the scheduling algorithm in terms of cooperative game theory which allows to quantify the impact of an organization on the system. We present a non-monetary model in which it is not required that each organization has accurate valuations of its jobs and resources. We show that classic utility functions may create incentives for workload manipulations. We thus propose a strategy resilient utility function that can be thought of as per-organization throughput.

We analyze the complexity of the fair scheduling problem. The general problem is NP-hard and difficult to approximate. Nevertheless, the problem parametrized with the number of organizations is FPT. Also, the FPT algorithm can be used as a reference for comparing the fairness of different algorithms on small instances. For a special case with unit-size jobs, we propose a FPRAS. In the full version of this paper [37] we show that the FPRAS can used as a heuristic algorithm; we also show another efficient heuristic. Our experimental evaluation indicates that the two algorithms produce reasonably fair schedules.

Since we do not require the valuation of the jobs, and we consider an on-line, non-clairvoyant scheduling, we believe the presented results have practical consequences for real-life job schedulers. In our future work we plan to use our fairness metric to experimentally assess standard scheduling algorithms, such as FCFS or fair-share. Also, we want to extend our model to parallel jobs.

**Acknowledgements** The research is funded by the Fundation for Polish Science "Homing Plus" Programme co-financed by the European Regional Development Fund (Innovative Economy Operational Programme 2007-2013). Piotr Skowron is partly supported by the European Union Human Capital Program "National PhD Programme in Mathematical Sciences" carried out at the University of Warsaw.

# 7. REFERENCES

[1] J. Bruno, J. Brustoloni, E. Gabber, B. Ozden, and A. Silberschatz. Disk scheduling with quality of service guarantees. In *ICMCS, Proc.*, page 400, 1999.

[2] R. Buyya, D. Abramson, and S. Venugopal. The grid economy. In *Special Issue on Grid Computing*, volume 93, pages 698–714, 2005.

[3] T. E. Carroll and D. Grosu. Divisible load scheduling: An approach using coalitional games. In *ISPDC, Proc.*, 2007.

[4] H. M. Chaskar and U. Madhow. Fair scheduling with tunable latency: a round-robin approach. *IEEE/ACM Trans. Netw.*, 11(4):592–601, 2003.

[5] G. Christodoulou and E. Koutsoupias. The price of anarchy of finite congestion games. In *STOC, Proc.*, pages 67–73, 2005.

[6] P.-F. Dutot, F. Pascual, K. Rzadca, and D. Trystram. Approximation algorithms for the multi-organization scheduling problem. *IEEE Transactions on Parallel and Distributed Systems*, 22:1888 – 1895, 2011.

[7] M. Ehrgott. Approximation algorithms for combinatorial multicriteria optimization problems. *International Transactions in Operational Research*, 7:2000, 2000.

[8] P. Ghosh, K. Basu, and S. K. Das. A game theory-based pricing strategy to support single/multiclass job allocation schemes for bandwidth-constrained distributed computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 18(3):289–306, 2007.

[9] P. Ghosh, N. Roy, S. K. Das, and K. Basu. A pricing strategy for job allocation in mobile grids using a non-cooperative bargaining theory framework. *Journal of Parallel and Distributed Computing*, 65(11):1366 – 1383, 2005.

[10] P. Goyal, H. M. Vin, and H. Chen. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. In *SIGCOMM, Proc.*, pages 157–168, 1996.

[11] D. Grosu and A. T. Chronopoulos. A game-theoretic model and algorithm for load balancing in distributed systems. In *IPDPS, Proc.*, pages 146–153, 2002.

[12] D. Grosu and A. T. Chronopoulos. A truthful mechanism for fair load balancing in distributed systems. In *NCA, Proc.*, 2003.

[13] D. Grosu and A. Das. Auction-based resource allocation protocols in grids. In *IPDCS, Proc.*, pages 20–27, 2004.

[14] A. Gulati and I. Ahmad. Towards distributed storage resource management using flow control. *SIGOPS Oper. Syst. Rev.*, 42(6):10–16, 2008.

[15] A. Gulati, I. Ahmad, and C. A. Waldspurger. PARDA: Proportional Allocation of Resources for Distributed Storage Access, Feb. 2009.

[16] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell. A survey of peer-to-peer storage techniques for distributed file systems. In *ITCC, Proc.*, pages 205–213, 2005.

[17] L. Huang, G. Peng, and T.-c. Chiueh. Multi-dimensional storage virtualization. *SIGMETRICS Perform. Eval. Rev.*, 32(1):14–24, 2004.

[18] A. Inoie, H. Kameda, and C. Touati. Pareto set, fairness, and nash equilibrium: A case study on load balancing. In *In Proc. of the 11th Intl. Symp. on Dynamic Games and Applications*, pages 386–393, 2004.

[19] D. B. Jackson, Q. Snell, and M. J. Clement. Core algorithms of the maui scheduler. In *JSSPP, Proc.*, pages 87–102, 2001.

[20] W. Jin, J. S. Chase, and J. Kaur. Interposed proportional sharing for a storage service utility. *SIGMETRICS Perform. Eval. Rev.*, 32(1):37–48, 2004.

[21] M. Karlsson, C. Karamanolis, and X. Zhu. Triage: Performance differentiation for storage systems using adaptive control. *Trans. Storage*, 1(4):457–480, 2005.

[22] C. Kenyon and G. Cheliotis. Grid resource commercialization: economic engineering and delivery scenarios. In *Grid resource management: state of the art and future trends*, pages 465–478. 2004.

[23] M. M. Kostreva, W. Ogryczak, and A. Wierzbicki. Equitable aggregations and multiple criteria analysis. *EJOR*, 158(2):362–377, 2004.

[24] C. Lee and A. Snavely. On the user–scheduler dialogue: studies of user-provided runtime estimates and utility functions. *International Journal of High Performance Computing Applications*, 20(4):495–506, 2006.

[25] D. Liben-Nowell, A. Sharp, T. Wexler, and K. Woods. Computing shapley value in supermodular coalitional games. In *COCOON, Proc.*, pages 568–579, 2012.

[26] L. Mashayekhy and D. Grosu. A merge-and-split mechanism for dynamic virtual organization formation in grids. In *PCCC, Proc.*, pages 1–8, 2011.

[27] D. Mishra and B. Rangarajan. Cost sharing in a job scheduling problem using the shapley value. In *EC, Proc.*, pages 232–239, 2005.

[28] H. Moulin. On scheduling fees to prevent merging, splitting, and transferring of jobs. *Math. Oper. Res.*, 32(2):266–283, May 2007.

[29] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*, chapter Routing Games. Cambridge University Press, 2007.

[30] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*, volume 1 of *MIT Press Books*. The MIT Press, 1994.

[31] S. Penmasta and A. T. Chronopoulos. Price-based user-optimal job allocation scheme for grid systems. In *IPDPS, Proc.*, pages 336–336, 2006.

[32] R. Rahman, T. Vinkó, D. Hales, J. Pouwelse, and H. Sips. Design space analysis for modeling incentives in distributed systems. In *SIGCOMM, Proc.*, pages 182–193, 2011.

[33] A. Roth. The price of malice in linear congestion games. In *WINE, Proc.*, pages 118–125, 2008.

[34] K. Rzadca, D. Trystram, and A. Wierzbicki. Fair game-theoretic resource management in dedicated grids. In *CCGRID, Proc.*, 2007.

[35] L. S. Shapley. A value for n-person games. *Contributions to the theory of games*, 2:307–317, 1953.

[36] P. Skowron and K. Rzadca. Network delay-aware load balancing in selfish and cooperative distributed systems. In *IPDPS Workshops, Proc.*, 2013.

[37] P. Skowron and K. Rzadca. Non-monetary fair scheduling — cooperative game theory approach. *CoRR*, abs/1302.0948, http://arxiv.org/pdf/1302.0948, 2013.

[38] B. Vocking. Selfish load balancing. In N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, editors, *Algorithmic Game Theory*. Cambridge University Press, 2007.

[39] K. D. Vohs, N. L. Mead, and M. R. Goode. The Psychological Consequences of Money. *Science*, 314(5802):1154–1156, Nov. 2006.

[40] Y. Wang and A. Merchant. Proportional-share scheduling for distributed storage systems, 2007.

[41] M. Welsh and D. Culler. Adaptive overload control for busy internet servers, 2003.

[42] A. Wierman. Fairness and scheduling in single server queues. *Surveys in Operations Research and Management Science*, 16:39–48, 2011.

[43] H. Yaïche, R. R. Mazumdar, and C. Rosenberg. A game theoretic framework for bandwidth allocation and pricing in broadband networks. *IEEE/ACM Trans. Netw.*, 8(5):667–678, Oct. 2000.

[44] J. Zhang, A. Sivasubramaniam, A. Riska, Q. Wang, and E. Riedel. An interposed 2-level i/o scheduling framework for performance virtualization. In *SIGMETRICS, Proc.*, pages 406–407, 2005.

[45] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin. What does control theory bring to systems research? *SIGOPS Oper. Syst. Rev.*, 43(1):62–69, 2009.