# A General Framework for Energy-Efficient Cloud Computing Mechanisms

Antonios Antoniadis * Andrés Cristi † Tim Oosterwijk ‡ Alkmini Sgouritsa §

**Abstract**

We present a general model for the operation of a cloud computing server comprised of one or more speed-scalable processors. Typically, tasks are submitted to such a cloud computing server in an online fashion, and the server operator has to schedule the tasks and decides on payments without knowledge about the tasks arriving in the future. Although very natural, this cloud computing problem on speed-scalable processors has not been studied from a mechanism design perspective in the online setting.

We provide a mechanism for this setting, both for a single and multiprocessor environment, that has several desirable properties: (1) the induced game admits a subgame perfect equilibrium in pure strategies and therefore a pure Nash equilibrium, (2) the Price of Anarchy is constant, (3) the mechanism is budget balanced, i.e., the sum of the payments of the agents is equal to the total energy costs, (4) the communication complexity is low, (5) the mechanism is computationally tractable for both the service operator and the agents, and (6) the agents' payment is also intuitive and easy to communicate to them. We also provide a second mechanism with a better Price of Anarchy, which in turn is more involved to implement.

We are able to extend our mechanisms and results to the Bayesian setting, where the type of each agent is drawn independently from some underlying distribution and agents are minimizing their expected costs. In this setting we also show the same approximation factor of our mechanism as in the basic online setting in both the single and the multiprocessor environment.

## 1 Introduction

Data centers hosting cloud services are enormous facilities, that typically may consume as much energy as 25,000 households [21]. During the last two decades, many such data centers have opened around the world in order to satisfy the continued growth of the IT industry, together with the increasing requirements for reliable data processing, data storage and on-demand computation. This growing infrastructure results in a growing energy consumption and environmental impact. In fact, data centers consumed in 2010 between 1% and 1.5% of the total electricity produced globally, and are estimated to use between 3% and 13% by 2030 [5]. As a result, energy-efficiency in data centers is a major concern not only from the operator perspective, but also for society in general.

With energy-efficiency in mind, one advantage of a cloud computing data center is that it allows to optimize many processes in a centralized manner. Research into this area can therefore

*Saarland University and Max-Planck-Institute for Informatics, Saarland University Campus, Email:aantonia@mpi-inf.mpg.de. This author was supported by DFG grant AN 1262/1-1.

†Universidad de Chile, Email:andres.cristi@ing.uchile.cl. This author was supported by CONICYT-PFCHA/Doctorado Nacional/2018-21180347.

‡Maastricht University, Email:t.oosterwijk@maastrichtuniversity.nl.

§University of Liverpool, Email:alkmini@liverpool.ac.uk. Part of this work was done at Max-Planck-Institute for Informatics, Saarland University Campus, Saarbrücken, Germany, supported by the Lise Meitner Award Fellowship.

have a crucial impact on the environmental effects of these centers. However, for this to be the case, one needs to take into account two important aspects when designing a model for such data centers. The first one is the online nature of the problem. A cloud computing service operates and receives tasks continuously, usually with little knowledge of upcoming future requests. The second important aspect is that there are multiple agents, with non-uniform priorities, requirements and information. The modeling approach should take into account their incentives in their relation with the cloud service. Therefore, it is natural to study the problem from a game-theoretic perspective and to find a mechanism for the processing that results in an equilibrium[1] with good performance guarantees.

This modeling approach has been followed in previous work. However, the main objective in mind has been to optimize the performance of the cloud server in terms of the total profit obtained from the processed tasks [10, 19, 27], and little has been done in terms of energy efficiency. Arguably, the most important factor regarding the energy expenditure in the day-to-day operation is the speed at which tasks are processed. The capability to dynamically adjust the speed of the processor(s) is a typical functionality implemented by chip manufacturers and is commonly referred to as *dynamic speed scaling*. While a higher speed implies a higher Quality of Service (QoS), it also comes with a higher energy consumption. A balance needs to be struck between these opposing objectives of performance and energy utilization.

As is common in the literature, we assume the energy consumption per unit of time in a given processor is of the form $s^\alpha$, where $s$ is the processing speed and $\alpha > 1$ is a fixed parameter. It turns out that, in practice, the energy consumption per unit of time is proportional to roughly the cube of the speed and thus $\alpha$ is usually assumed to be between 2 and 3 [17, 30]. The total energy consumed by the processor is the integral of this function over time.

We study the operation of a cloud computing data center, modeled as a system of shared processors with speed scaling capabilities. Agents of the system arrive in an online manner, each with a job of a certain workload that needs to be processed by the server. Each agent communicates information about her job to the service operator, gets her job returned to her after it is completed, and is charged a payment for the processing. The service operator is concerned with the total energy expended, together with the QoS that each agent perceives, which is modeled as a private cost function that is non-decreasing in the waiting time. In particular we assume that the operator wants to minimize the total energy spent plus the sum of all waiting costs, while guaranteeing that the payments cover the energy expenditure; we consider that each agent tries to minimize the sum of her payment and her waiting cost.

We are interested in the problem from the perspective of the service operator, who is in charge of implementing some mechanism to process the jobs. The service operator receives the information about the job characteristics (scheduling time constraints and workload) of each agent and decides on a feasible processing schedule based on this communication. This schedule specifies for each time point which job is running on each processor, and at what speed.

After the server operator runs the schedule he computes, he returns each job upon completion, and receives a payment from each agent. The amount to be paid is also to be determined by the service operator based on the mechanism he implements. We consider the agents to be strategic, so they submit their job's characteristics under the goal of minimizing their personal cost which is their payment and waiting cost. Therefore, we assume that the outcome of a given mechanism is the equilibrium that results of the interaction between the agents and the server. The quality of this outcome is measured as the sum of the energy cost and the waiting costs of all agents, also

[1] An equilibrium is a stable situation in which no agent can be better off by making another decision, given the choices of the other agents of the system.

referred to as the social cost of the equilibrium. The goal is to ensure that this is not too far off from the optimum.[2] Thus, we search for a mechanism that results in a low *Price of Anarchy* (PoA) (introduced by Koutsoupias and Papadimitriou in 1999 [25]), which is the worst case ratio between the social costs of any equilibrium and the social cost of the optimum.

Recently, Antoniadis and Cristi [7] studied the version of the problem in which agents arrive simultaneously in the system. They presented the modAVR mechanism, which guarantees a constant PoA of $O(\alpha^{2\alpha-2})$ for a shared speed scalable processor with the same social cost function as ours. We extend their results to the online setting and to multiple processors. Our mechanism attributes the costs in a different way in the online setting and in order to deal with the multiprocessor setting we introduce novel techniques. Moreover, we extend all the results to the Bayesian case.

### 1.1 Our Contribution

We present a general model of the operation of a cloud computing server in Section 2, where we allow for agents to have arbitrary non-decreasing waiting cost functions and online arrivals, and where the server is comprised of multiple speed-scalable processors. Even though our model does not capture every single aspect in the operation of a real-world cloud server, it does capture the essential features: energy efficiency versus performance trade-off, multiplicity of incentives and online decision making. Thus, we have a clean mathematical formulation that allows us to study the problem from a theoretical point of view.

We provide a mechanism for this setting that has several desirable properties.

1. The induced game has a subgame perfect equilibrium in pure strategies and therefore a pure Nash equilibrium[3], i.e., a strategy profile in which every agent chooses a pure strategy (as opposed to a randomized strategy) that has the lowest costs given the pure strategies of the other agents.

2. We prove that the PoA is constant for any fixed value of $\alpha$. In other words, the total costs are not far off from the social optimum.

3. The mechanism is budget balanced, that is to say, the sum of the payments of the agents is equal to the total energy costs.

4. The communication complexity is low, as agents need not declare their full private waiting cost function but only the starting time, the completion time and the workload of their job.

5. The mechanism is computationally tractable for both the service operator and the agents: when an agent arrives, she can efficiently compute her optimal strategy and the operator can efficiently compute the processing schedule and the payment of the agent.

6. The payment of the agents is also intuitive and easy to explain to them.

Our mechanism is called $\text{IncrAVR}_{\text{SW}}$ and it combines the Incremental cost-sharing protocol [29] with the AVR (average rate) algorithm [33] for scheduling jobs. The index SW is an abbreviation for Single Window and indicates that agents can declare only a single time interval in which their job needs to be processed.

[2] We consider the optimum to be the feasible schedule that minimizes the social cost.

[3] In the offline setting, Antoniadis and Cristi [7] only showed that under the modAVR mixed Nash equlibria are guaranteed to exist, and furthermore it is not clear whether computing an equilibrium is computationally tractable.

We additionally present another multi window mechanism called $\text{IncrAVR}_{\text{MW}}$, which differs from the single window variant in that the agents can now submit a union of (disjoint) time intervals in their strategy. The Price of Anarchy we prove for this more general mechanism is better, but the single window variant has the desirable properties 4 and 5 above regarding its simplicity, both in computation and in communication.

The energy cost of the solution found by AVR is bounded by $2^{\alpha-1}\alpha^{\alpha}$ times the energy costs of the optimal schedule. In our online setting, where agents do not only pay for the energy costs but also have their private cost function measuring their perceived QoS, we manage to find the same PoA for $\text{IncrAVR}_{\text{MW}}$ (cf. Theorem 3.5). On the other hand, our simpler mechanism $\text{IncrAVR}_{\text{SW}}$ only loses the same constant factor another time (cf. Theorem 3.6).

Furthermore, we extend the results to the case with multiple processors. In this case, we present a simple greedy algorithm to match jobs with processors, whose energy consumption is bounded by a factor of $2^{2\alpha}$ times the energy consumption of the AVR algorithm in the single window multiprocessor setting or that of the optimum schedule in the multi window multiprocessor setting. Thus, we can show that the mechanism induced by the greedy algorithm has a PoA bounded by $2^{3\alpha-1}\alpha^{\alpha}$ in the multi window case and by roughly $2^{4\alpha-2}\alpha^{2\alpha}$ in the single window case (cf. Theorems 4.3 and 4.4). We defer a more complicated algorithm for the multiprocessor case that achieves a PoA improved by a factor $2^{2\alpha}$ to the full version.

Finally, we extend our mechanisms and results to the Bayesian setting, where the type of each agent is drawn independently from some underlying distribution and agents are minimizing their expected costs. In this setting we also show the same approximation factor of our mechanisms as in the basic online setting (cf. Corollaries 5.4, 5.5, 5.6 and 5.7).

Our main approach for proving the PoA upper bounds is by applying the smoothness technique of Roughgarden's work [31] through a technical lemma that we prove. Regarding the Bayesian setting, we derive the same bounds by using the Extension Theorem of the same paper.

While other mechanisms might be more obvious to consider, they do not possess all the properties our mechanism achieves. Consider for example the mechanism that recomputes the social optimum upon the arrival of a new task, i.e., it processes the remaining jobs in a way that minimizes the overall costs of the system, given what has already been processed before. The computational overhead of this mechanism for both the service operator and the agents prevents this mechanism from being deployed in a practical setting[4]. Moreover, if we consider proportional cost sharing, the cost of an agent depends on the strategies of agents arriving into the system after this agent, and it is unclear whether pure equilibria exist.

### 1.2 Related Work

There is a large body of literature related to dynamic speed scaling. This was initiated by the seminal paper by Yao, Demers and Shenkers [33], who presented an optimal algorithm to solve deadline based speed scaling problems. Their algorithm, which is now usually referred to as the YDS-algorithm, minimizes the energy consumption while meeting the deadlines, and it was formally analyzed by Bansal et al. [14]. The AVR algorithm, on which our work is partially based, was also first proposed by Yao et al. for the online version of the problem, and has a constant competitive ratio [11]. Our results for the multiprocessor setting are somewhat inspired by the extension of AVR to the multiple machine setting, that also has a constant competitive ratio [3].

Several other variants of the problem have been studied as well, e.g. the case when processors have an upper bound on their speeds [13] or are equipped with a sleep-state [2, 8]. Also different

[4]The fastest algorithm that computes an optimal schedule takes time cubic in the number of tasks, and each agent would have to run it on every possible combination of strategies of the upcoming agents to make a decision.

objectives have been studied, such as throughput maximization [6] and the combination of energy costs plus flow time [4, 26, 18, 15]. The latter objective resembles some relevance to our setting as the flow time can be simulated by linear waiting cost functions. We refer to [1, 24] for two surveys on dynamic speed scaling.

Previous literature regarding scheduling in cloud services has focused on the problem of maximizing the value of tasks completed before their deadlines, using a processor with constant speed. Lucier et al. [27] showed that if there is always some slack to delay a job and still finish it by its deadline, there is an algorithm that achieves a constant-factor worst-case guarantee. They also designed a truthful mechanism based on their algorithm. In a different direction, Chawla et al. [19] studied the case of stochastic job arrivals, and designed a truthful mechanism with near-optimal performance. Recently, in an effort to bring theoretical work in cloud scheduling closer to real-world applications, Babaioff et al. [10] proposed the Economic Resource Allocation (ERA) model and framework, with a modular structure that receives requests from agents that specifies an amount of resources, a time window and a will to pay; a pluggable algorithm that computes a schedule, and an interface to communicate with the cloud.

Prior to the work of Antoniadis and Cristi [7], two other papers investigated speed scaling scheduling from a game theoretic perspective. Dürr, Jez and Vasquez [23] studied the existence of pure Nash equilibria under two different offline mechanisms based on the YDS algorithm. In particular, they showed that charging the agents proportionally does not necessarily admit a pure Nash equilibrium whereas charging the marginal costs does, however at the expense of overcharging the agents. It is unclear if any of their mechanisms induces a bounded PoA. The same set of authors also designed a truthful and budget balanced mechanism in which the scheduler fixes an arbitrary order of the jobs before the agents declare their waiting cost functions [22]. However, considering a fixed order for the tasks is a very strong constraint which can lead to an arbitrarily high PoA [7].

Also related to our work is the study of the underlying offline optimization problem. There exists a PTAS for minimizing weighted flowtime in a single machine with energy budget and a $(2+\varepsilon)$-approximation for the case of multiple machines [28]. The weighted flowtime can be seen as the sum of the waiting costs for the case of linear functions. Interestingly, it remains an open question whether the problem is NP-hard.

Finally, several variants energy-efficiency in a cloud computing setting have received attention in the autonomous agents and multi-agent systems community. See [16] for a survey and [32, 20] for some examples.

## 2 Model

Consider a cloud computing service with $n$ agents indexed by the elements $i \in [n] = \{1, \ldots, n\}$. Agent $i$ arrives at time $a_i \geq 0$ and has a job with *workload* $w_i > 0$ that needs to be processed. Each job may be processed in several intervals of time. We assume that the time is split into time units that represent the minimum length of time that any job is processed without interruption; therefore the length of the aforementioned intervals should be multiples of the time unit.[5]

We restrict ourselves to mechanisms where the strategy profile $\boldsymbol{\tau}$ is a set of tuples $\tau_i = (I_i, w_i)$ for each $i \in [n]$, where $I_i$ represents a union of (disjoint) intervals during which the job of agent $i$ should be scheduled; naturally, $I_i \subset [a_i, \infty)$, that is, no interval should begin before the arrival of agent $i$'s job.

[5]We remark that time discretization is not really needed in order to show our results. It is rather a technicality to keep our presentation simple and not to consider zero length intervals separately.

Then, a mechanism decides on a schedule to process the jobs by defining a "virtual" speed $s_i(t)$ for each agent $i$ and time point $t \in I_i$ (we assume $s_i(t) = 0$ for $t \notin I_i$). We consider virtual speeds by allowing more than two jobs to be processed at the same time, i.e., there can exist $i, j \in [n]$ and $t$ such that both $s_i(t) > 0$ and $s_j(t) > 0$. In fact, the jobs do not run in parallel, but at each time point a single job is processed at the aggregated speed. Since the outcome of this schedule would be the same as the schedule of parallel processing at the virtual speeds, we consider the latter in order to keep the presentation simple. We do require the speeds to be feasible, meaning that the whole workload of agent $i$ needs to be fully processed during $I_i$, i.e.,

$$\int_{I_i} s_i(t)\,\mathrm{d}t = w_i\,.$$

The energy cost of executing the schedule $S$ in a processor depends on a fixed parameter $\alpha > 1$ and is given by

$$\mathcal{E}(S) = \int_0^\infty \left(\sum_{i:t\in I_i} s_i(t)\right)^\alpha \mathrm{d}t\,.$$

For a given schedule, each agent $i \in [n]$ incurs a (waiting) cost $f_i(c_i)$, where $c_i$ is the completion time of his job. This function models the Quality of Service agent $i$ perceives and is non-decreasing and private.

A service operator determines a mechanism to communicate with the agents, computes a schedule $S$ based on the strategy profile $\boldsymbol{\tau}$ and requests a payment $\phi_i(\boldsymbol{\tau})$ from each agent $i \in [n]$. We say that the mechanism is *budget balanced* if $\sum_{i=1}^n \phi_i(\boldsymbol{\tau}) = \mathcal{E}(S)$. Each agent $i$ seeks to minimize her costs $C_i(\boldsymbol{\tau})$, which is the sum of her waiting cost and the payment, i.e.,

$$C_i(\boldsymbol{\tau}) = f_i(c_i) + \phi_i(\boldsymbol{\tau})\,. \tag{1}$$

For notational convenience we let $C(\boldsymbol{\tau}) = \sum_i C_i(\boldsymbol{\tau}), f(\mathbf{c}) = \sum_i f_i(c_i)$ and $\phi(\boldsymbol{\tau}) = \sum_i \phi_i(\boldsymbol{\tau})$, where $\mathbf{c} = (c_1, \ldots, c_n)$ is the vector of completion times.

**IncrAVR$_{\mathrm{MW}}$.** Next to some mechanisms that are already described in the literature, we introduce a new mechanism in this paper that we call *IncrAVR$_{MW}$* and it uses the AVR algorithm [33] to decide the schedule and an incremental scheme [29] in order to define the payments.

In the IncrAVR$_{\mathrm{MW}}$ mechanism, each agent $i \in [n]$ submits the values $\tau_i = (I_i, w_i)$ to the service operator upon arrival. The service operator computes the speed of each agent $i$ according to the AVR algorithm, i.e.,

$$s_i(t) = \frac{w_i}{|I_i|} \text{ for } t \in I_i \quad \text{ and } \quad s_i(t) = 0 \text{ otherwise}, \tag{2}$$

and the completion time of agent $i$'s job, i.e. $c_i$, is the end of $I_i$ (by using the same approach as in [7]).

Moreover, IncrAVR$_{\mathrm{MW}}$ initially defines a global ordering $\pi$ of the agents. Starting with an empty schedule $S_0$ and considering the agents one after the other based on the ordering $\pi$, the service operator creates schedule $S_i$ by adding agent $i$'s job the to the existing schedule $S_{i-1}$ as described above. Each agent $i$ is charged the difference between the energy cost of the previous schedule $S_{i-1}$ with the $i-1$ first agents and the new schedule $S_i$ with agent $i$ included, i.e., $\phi_i(\boldsymbol{\tau}) = \phi_i(\tau_1, \ldots, \tau_i) = \mathcal{E}(S_i) - \mathcal{E}(S_{i-1})$.

**IncrAVR$_{\mathbf{SW}}$.** We further suggest *IncrAVR$_{SW}$* which is similar to IncrAVR$_{\mathrm{MW}}$; the only difference is that the IncrAVR$_{\mathrm{SW}}$ restricts the strategy space of the agents and allows them to declare only a single interval, i.e. here $I_i$ is a single interval. This restriction will obviously result in different equilibrium outcomes; it is not clear if the quality of those equilibria is better or worse with respect to the ones induced by IncrAVR$_{\mathrm{MW}}$; though we could prove better guarantees for IncrAVR$_{\mathrm{MW}}$. The most important advantage of IncrAVR$_{\mathrm{SW}}$ against IncrAVR$_{\mathrm{MW}}$ is the simplicity; agents should simply communicate a single interval and it is easier from their perspective to decide which one minimizes their own cost.

We remark that both IncrAVR$_{\mathrm{MW}}$ and IncrAVR$_{\mathrm{SW}}$ can be used in both offline and online versions of this problem. In the offline setting, $\pi$ can be an arbitrary order of all agents, whereas in the online setting, $\pi$ should be defined according to the arrival order of the agents.

**Remark 2.1.** Note that if the agents iteratively (following $\pi$) choose a pure best response by knowing only the strategies of all agents that precede them, this would result in a subgame perfect equilibrium. This is because under both our mechanisms the agents' cost are *not* affected by the choices of the following agents in $\pi$. Therefore, a pure (full information or Bayesian) Nash equilibrium always exists in our mechanism which is also a subgame perfect equilibrium.

## 2.1 Preliminaries

Each agent $i$ has a type $t_i = (a_i, w_i, f_i)$ consisting of three parameters: her arrival time, the workload of her job and a private (waiting) cost function of the completion time of the job. In the full information setting, $t_i$ is fixed for each agent $i$, and therefore for notational convenience we drop the dependency on $t_i$.

A feasible action of agent $i$, $\tau_i = (I_i, \bar{w}_i)$, consists of two parameters, a union of intervals $I_i$ and the declared workload $\bar{w}_i$. By $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)$ we denote the strategy profile and $\boldsymbol{\tau}_{-i}$ denotes the strategies of all agents but $i$. We say that the strategy is *feasible* if it satisfies that $I_i \subset [a_i, \infty)$ and $\bar{w}_i \geq w_i$. The agent may strategically decide to process her job with some delays or to submit an increased workload by adding more processing requirements, if she believes that this could lead to a reduction of her total cost.

We remark that in the IncrAVR$_{\mathrm{MW}}$ (and in the IncrAVR$_{\mathrm{SW}}$) mechanism, no agent has an incentive to declare a different workload, so in the rest of the paper we make no difference between $w_i$ and $\bar{w}_i$. This is because misreporting it wouldn't affect the agents that precede her in $\pi$ (the arrival order in the online setting) and given the actions of those agents, her payment increases with $\bar{w}_i$.

**Online Setting.** In the online version (still assuming the full information setting) each agent $i$ arrives at time $a_i$ and needs to decide on $\tau_i$ while she only has knowledge of the schedule up to $a_i$ (or more realistically, the future load due to arrivals so far). We name the agents based on their arrival order (resolving ties arbitrarily), i.e., agent $i$ is the agent with the $i$'th smallest arrival time. Therefore, agent $i$ chooses $\tau_i$ by knowing the schedule of agents $j < i$.

Recall that IncrAVR$_{\mathrm{MW}}$ (and IncrAVR$_{\mathrm{SW}}$) uses the arrival order $\pi$. This means that the payment of each agent depends only on the agents that have already declared their strategies and not on the agents later in the sequence. Therefore, each agent can compute her best response upon arrival. Hence, it is immediate that the resulting strategy profile $\boldsymbol{\tau}$ is a (subgame perfect) Nash equilibrium, i.e., for any agent $i$ and any alternative feasible strategy $\tau_i'$ for $i$, we have

$$C_i(\boldsymbol{\tau}) \leq C_i(\tau_i', \boldsymbol{\tau}_{-i})\,.$$

**Bayesian Setting.** In the Bayesian setting, each agent $i$ has a type space $\mathcal{T}_i$ and as before, her type $t_i \in \mathcal{T}_i$ is the tuple $t_i = (a_i, w_i, f_i)$. Given $t_i$ we denote the first and the second argument of $t_i$ by $a_i(t_i)$ and $w_i(t_i)$, respectively. A type vector for all agents is $\mathbf{t} = (t_1, \ldots, t_n) \in \mathbf{T} = \times_i \mathcal{T}_i$ and by $\mathbf{t}_{-i}$ we denote the types of all agents but $i$. The type of each agent is drawn from $\mathcal{T}_i$ according to some distribution $\mathcal{F}_i$, which is common knowledge. We assume that the types $t_i$ are *independent* (but not necessarily identically distributed). Thus, the type vector is drawn from $\mathbf{T}$ according to the product distribution, $\mathbf{F} = \prod_i \mathcal{F}_i$. If $\mathbf{F}$ is a mass point, then the game is equivalent to a full information game.

Each agent is aware only of her own type $t_i$ and of the product distribution $\mathbf{F}$ (not the realization of their types). Therefore, agent $i$ decides a strategy $\sigma_i(t_i)$ for every type $t_i \in \mathcal{T}_i$. $\sigma_i(t_i)$ is in general a mixed strategy drawn from some action space $\mathcal{A}_i$, i.e., a probability distribution over pure strategies from $\mathcal{A}_i$. In the special case that agent $i$ decides a pure strategy, then this is a single value $\tau_i(t_i) \in \mathcal{A}_i$. Let $\mathbf{A} = \times_i \mathcal{A}_i$. By $\boldsymbol{\tau}(\mathbf{t}) = (\tau_1(t_1), \ldots, \tau_n(t_n))$ we denote the pure strategies of all agents and as usual $\boldsymbol{\tau}_{-i}(\mathbf{t})$ are the strategies of all agents but $i$. Similarly we use $\boldsymbol{\sigma}(\mathbf{t}) = (\sigma_1(t_1), \ldots, \sigma_n(t_n))$ and $\boldsymbol{\sigma}_{-i}(\mathbf{t})$ for mixed strategies.

The cost of agent $i$ (given in Equation (1)), depends on her type $t_i$ and on the strategy profile $\boldsymbol{\tau}$, i.e., we write $C_i(t_i; \boldsymbol{\tau})$. In a Bayesian Nash equilibrium, each agent $i$ selects a (mixed) strategy, that they do not want to deviate from, by knowing only their own type $t_i$, $\mathbf{F}$ and the strategies of the other agents $\boldsymbol{\sigma}_{-i}$. Formally, a strategy profile $\boldsymbol{\sigma}$ is a Bayesian Nash equilibrium if for every agent $i$, type $t_i \in \mathcal{T}_i$, and feasible strategy $\tau_i'$ for $t_i$,

$$\begin{aligned} &\mathbb{E}_{\mathbf{t}_{-i} \sim \mathbf{F}_{-i}} \left[ \mathbb{E}_{\boldsymbol{\tau} \sim \boldsymbol{\sigma}(\mathbf{t})} [C_i(t_i; \boldsymbol{\tau})] \right] \\ \leq &\mathbb{E}_{\mathbf{t}_{-i} \sim \mathbf{F}_{-i}} \left[ \mathbb{E}_{\boldsymbol{\tau}_{-i} \sim \boldsymbol{\sigma}_{-i}(\mathbf{t}_{-i})} \left[ C_i(t_i; (\tau_i', \boldsymbol{\tau}_{-i})) \right] \right] . \end{aligned}$$

**Multiprocessor Setting.** In the multiprocessor setting, we can distribute the jobs to $m$ processors in order to keep speeds lower and incur a lower energy consumption. Although a job can be assigned to more than one processors, we assume that jobs are sequential and therefore cannot be processed on two different processors at the same timepoint[6]. The energy cost of a schedule now is the sum of the energy costs of the individual processors in the schedule.

**Price of Anarchy.** To measure the efficiency of a strategy profile $\boldsymbol{\tau}$ under a mechanism $M$, we use the total cost $C_M(\boldsymbol{\tau}) = f(\mathbf{c}) + \phi_M(\boldsymbol{\tau})$,[7] where we drop the subscript if it is clear from the context. We quantify the performance of $M$ by using the Price of Anarchy (PoA) metric introduced in [25]. We consider as the social optimum the minimum possible social cost under any feasible schedule.

As mentioned earlier, given a workload vector $\mathbf{w} = (w_1, \ldots, w_n)$, a release vector $\mathbf{r} = (r_1, \ldots, r_n)$ and a deadline vector $\mathbf{d} = (d_1, \ldots, d_n)$, the YDS algorithm [33] results in the schedule of minimum energy cost (see [14] for its formal analysis), where each workload $w_i$ is scheduled between $r_i$ and $d_i$ (assuming $d_i > r_i$). Let $\mathcal{E}_{\text{YDS}}(\mathbf{r}, \mathbf{d}, \mathbf{w})$ be the energy consumption of the YDS algorithm. Given an arrival vector $\mathbf{a} \leq \mathbf{r}$ (where the inequality is pairwise), note that $\mathcal{E}_{\text{YDS}}(\mathbf{a}, \mathbf{d}, \mathbf{w}) \leq \mathcal{E}_{\text{YDS}}(\mathbf{r}, \mathbf{d}, \mathbf{w})$. Then the optimum social cost given a type vector $\mathbf{t}$ is

$$C^*(\mathbf{t}) = \min_{\mathbf{d} \geq \mathbf{a}} (f(\mathbf{d}) + \mathcal{E}_{\text{YDS}}(\mathbf{a}, \mathbf{d}, \mathbf{w})) .$$

[6]We remark that this assumption is quite general and covers the easier case where the same job may run in many processors at the same time.

[7]Note that the $f_i$ functions depend only on the completion time.

We will compare $C(\cdot)$ with $C^*(\cdot)$. If $\mathrm{NE}(\mathbf{t})$ are the sets of all (Bayesian) Nash equilibria for the game induced by the type vector $\mathbf{t}$, we define the *Price of Anarchy* (PoA) of IncrAVR as follows:

$$\mathrm{PoA} = \sup_{\mathbf{t}} \frac{\max_{\boldsymbol{\tau} \in \mathrm{NE}(\mathbf{t})} C(\boldsymbol{\tau})}{C^*(\mathbf{t})} ,$$

where the supremum is taken over all possible type vectors.

# 3 Online Setting

In this section we show that in the online setting the PoA of $\mathrm{IncrAVR}_{\mathrm{MW}}$ and $\mathrm{IncrAVR}_{\mathrm{SW}}$ are at most $\alpha^\alpha 2^{\alpha-1}$ and $\left(\alpha^\alpha 2^{\alpha-1}\right)^2$, respectively, and at least $\alpha^\alpha$ for both mechanisms. We note that the dependency on $\alpha$ is necessary since that there exists an online instance on which *any* mechanism has a PoA of at least $e^{\alpha-1}/\alpha$. The lower bound instance is inspired by Bansal et al. [12] who generate an instance for deadline-based speed scaling, on which no online algorithm can have a competitive ratio better than $e^{\alpha-1}/\alpha$ for the energy consumption. The complete derivation can be found in the full version of the paper. We show the upper bounds by using the well-known smoothness technique of [31]. We start with the definition of *Smooth Games*. We note that all the following lemmas hold for both mechanisms and the proofs differ only at the end.

**Definition 3.1.** (Smooth Game [31]). *A cost-minimization game is $(\lambda, \mu)$-smooth with respect to a strategy profile $\boldsymbol{\tau}^*$ and a minimization objective $C$, if for every strategy profile $\boldsymbol{\tau}$,*

$$\sum_{i=1}^{n} C_i(\tau_i^*, \boldsymbol{\tau}_{-i}) \leq \lambda C(\boldsymbol{\tau}^*) + \mu C(\boldsymbol{\tau}) .$$

We will show that the online game induced by either $\mathrm{IncrAVR}_{\mathrm{MW}}$ or $\mathrm{IncrAVR}_{\mathrm{SW}}$ is $(\lambda, \mu)$-smooth for

$$\mu \in (0,1) ; \qquad \lambda = \frac{(1+\mu)}{\left((1+\mu)^{\frac{1}{\alpha-1}} - 1\right)^{\alpha-1}} . \tag{3}$$

We start by proving a useful technical lemma.

**Lemma 3.2.** *Let $y_1, \ldots, y_n > 0$, $X > 0$ and define $Y = \sum_{i=1}^n y_i$. Then for any $\alpha > 1$, $\mu \in (0,1)$ and $\lambda$ as in Equation (3), we have that*

$$\sum_{i=1}^{n} \left((y_i + X)^\alpha - X^\alpha\right) \leq \lambda Y^\alpha + \mu X^\alpha . \tag{4}$$

*Proof.* By convexity it holds that $(y_i + X)^\alpha + (y_j + X)^\alpha \leq (y_i + y_j + X)^\alpha + X^\alpha$ for $\alpha > 1$, so

$$\sum_{i=1}^{n} \left((y_i + X)^\alpha - X^\alpha\right) \leq (Y + X)^\alpha - X^\alpha .$$

Thus, if we define $Z = X/Y$, it suffices to show that $(1+Z)^\alpha - (1+\mu)Z^\alpha \leq \lambda$. It is easy to verify that $Z^* = \left((1+\mu)^{\frac{1}{\alpha-1}} - 1\right)^{-1}$ maximizes the left-hand side of the previous inequality. Therefore,

$$\begin{aligned}\lambda &= \left(1 + \frac{1}{(1+\mu)^{\frac{1}{\alpha-1}} - 1}\right)^\alpha - \frac{(1+\mu)}{\left((1+\mu)^{\frac{1}{\alpha-1}} - 1\right)^\alpha} \\ &= \frac{(1+\mu)}{\left((1+\mu)^{\frac{1}{\alpha-1}} - 1\right)^{\alpha-1}}\,. \end{aligned}$$

□

In order to keep the presentation simple, we show as an intermediate step that the game (induced by any of the two mechanisms) ignoring the waiting cost functions $f_i$ is $(\lambda, \mu)$-smooth for the values of Equation (3). Then, we use that result to show that the original game (considering both the waiting costs and the energy charges) is $(\lambda, \mu)$-smooth. Recall that for any feasible strategy profile $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)$, agent $i$ has to pay for the energy

$$\phi_i(\boldsymbol{\tau}) = \int_{I_i} \left( \left( \sum_{j \leq i : t \in I_j} s_j \right)^\alpha - \left( \sum_{j < i : t \in I_j} s_j \right)^\alpha \right) \mathrm{d}t\,, \tag{5}$$

where $s_j = w_j / |I_j|$.

**Lemma 3.3.** *For any pair of feasible strategy profiles $\boldsymbol{\tau}$, $\boldsymbol{\tau}^*$ and $\lambda, \mu$ as in Equation (3), it holds that*

$$\sum_{i=1}^n \phi_i(\tau_i^*, \boldsymbol{\tau}_{-i}) \leq \lambda \phi(\boldsymbol{\tau}^*) + \mu \phi(\boldsymbol{\tau})\,.$$

*Proof.* For any strategy profiles $\boldsymbol{\tau}$ and $\boldsymbol{\tau}^*$, we have that

$$\begin{aligned}\sum_{i=1}^n \phi_i(\tau_i^*, \boldsymbol{\tau}_{-i}) &= \sum_{i=1}^n \int_{I_i^*} \left( \left( s_i^* + \sum_{j < i : t \in I_j} s_j \right)^\alpha - \left( \sum_{j < i : t \in I_j} s_j \right)^\alpha \right) \mathrm{d}t \\ &\leq \int_0^\infty \sum_{i : t \in I_i^*} \left( \left( s_i^* + \sum_{j : t \in I_j} s_j \right)^\alpha - \left( \sum_{j : t \in I_j} s_j \right)^\alpha \right) \mathrm{d}t\,,\end{aligned}$$

where the inequality comes from the fact that the function $(y+x)^\alpha - x^\alpha$ is non-decreasing in $x$, when $\alpha > 1$. Now we can apply Lemma 3.2 for values of $\lambda, \mu$ according to Equation (3) to obtain

$$\begin{aligned}\sum_{i=1}^n \phi_i(\tau_i^*, \boldsymbol{\tau}_{-i}) &\leq \int_0^\infty \left( \lambda \left( \sum_{i : t \in I_i^*} s_i^* \right)^\alpha + \mu \left( \sum_{j : t \in I_j} s_j \right)^\alpha \right) \mathrm{d}t \\ &= \lambda \phi(\boldsymbol{\tau}^*) + \mu \phi(\boldsymbol{\tau})\,.\end{aligned}$$

□

Smoothness for the original game follows easily from Lemma 3.3.

**Lemma 3.4.** *For any pair of feasible strategy profiles $\boldsymbol{\tau}$, $\boldsymbol{\tau}^*$ and $\lambda, \mu$ as in Equation (3),*

$$\sum_{i=1}^n C_i(\tau_i^*, \boldsymbol{\tau}_{-i}) \leq \lambda C(\boldsymbol{\tau}^*) + \mu C(\boldsymbol{\tau})\,.$$

*Proof.* For any strategy profiles $\boldsymbol{\tau}$ and $\boldsymbol{\tau}^*$, it holds that

$$\begin{aligned}\sum_{i=1}^{n} C_i(\tau_i^*, \boldsymbol{\tau}_{-i}) &= \sum_{i=1}^{n} f_i(c_i^*) + \sum_{i=1}^{n} \phi_i(\tau_i^*, \boldsymbol{\tau}_{-i}) \\ &\leq f(\mathbf{c}^*) + \lambda\phi(\boldsymbol{\tau}^*) + \mu\phi(\boldsymbol{\tau}) \\ &\leq \lambda f(\mathbf{c}^*) + \mu f(\mathbf{c}) + \lambda\phi(\boldsymbol{\tau}^*) + \mu\phi(\boldsymbol{\tau}) \\ &= \lambda C(\boldsymbol{\tau}^*) + \mu C(\boldsymbol{\tau})\,,\end{aligned}$$

where the first inequality is due to Lemma 3.3 for the $\lambda, \mu$ of Equation (3) and the second inequality holds because $\lambda > 1$ for $\alpha > 1$. □

We are now ready to show the two bounds on the PoA.

**Theorem 3.5.** *The online game induced by IncrAVR$_{MW}$ has PoA $\leq \alpha^\alpha 2^{\alpha-1}$.*

*Proof.* Let $\boldsymbol{\tau}$ be any Nash equilibrium and $S^*$ be the schedule that minimizes the social cost. Let $I_i^*$ be the union of intervals at which agent $i$'s job is processed under $S^*$. Note that this means that for any two agents $i, j$, $I_i^*$ and $I_j^*$ are disjoint (in any schedule a single job is processed at each time unit) and for any $t \in I_i^*$, the speed is $w_i/|I_i^*|$ (the energy cost function is convex and therefore minimized if the workload is equally distributed along $I_i^*$). For $\tau_i^* = (I_i^*, w_i)$, the minimum social cost is given by $C(\boldsymbol{\tau}^*)$.

By the definition of Nash equilibrium and smoothness, we have

$$C(\boldsymbol{\tau}) = \sum_{i=1}^{n} C_i(\boldsymbol{\tau}) \leq \sum_{i=1}^{n} C_i(\tau_i^*, \boldsymbol{\tau}_{-i}) \leq \lambda C(\boldsymbol{\tau}^*) + \mu C(\boldsymbol{\tau})\,.$$

By rearranging the terms we get PoA $\leq \lambda/(1-\mu)$. Since we proved smoothness (Lemma 3.4) for $\lambda, \mu$ given in Equation (3), the ratio $\lambda/(1-\mu)$ is minimized for $\mu = 2^{\frac{\alpha-1}{\alpha}} - 1$, which gives

$$\text{PoA} \leq \frac{1}{(2^{1/\alpha}-1)^\alpha} \leq \alpha^\alpha 2^{\alpha-1}\,.$$ □

**Theorem 3.6.** *The online game induced by IncrAVR$_{SW}$ has PoA $\leq \alpha^{2\alpha} 2^{2\alpha-2}$.*

*Proof.* Let $\boldsymbol{\tau}$ be any Nash equilibrium and $\boldsymbol{\tau}^*$ be the feasible strategy profile resulting in the optimum schedule (this is, $\boldsymbol{\tau}^* \in \arg\min_{\boldsymbol{\tau}'}(f(\boldsymbol{\tau}') + \mathcal{E}_{\text{YDS}}(\boldsymbol{\tau}'))$). Then the minimum social cost is $C^* = f(\boldsymbol{\tau}^*) + \mathcal{E}_{\text{YDS}}(\boldsymbol{\tau}^*)$. The key difference from the proof of Theorem 3.5 is that $C(\boldsymbol{\tau}^*)$ is not the social optimum, because the optimal scheduling may not process each job in a single interval. We rather need to use the competitive ratio of the AVR algorithm [33, 11] and get $\mathcal{E}_{\text{IncrAVR}_{\text{SW}}}(\boldsymbol{\tau}^*) = \mathcal{E}_{\text{AVR}}(\boldsymbol{\tau}^*) \leq \alpha^\alpha 2^{\alpha-1}\mathcal{E}_{\text{YDS}}(\boldsymbol{\tau}^*)$. As before,

$$\begin{aligned}C(\boldsymbol{\tau}) &= \sum_{i=1}^{n} C_i(\boldsymbol{\tau}) \leq \sum_{i=1}^{n} C_i(\tau_i^*, \boldsymbol{\tau}_{-i}) \leq \lambda C(\boldsymbol{\tau}^*) + \mu C(\boldsymbol{\tau}) \\ &\leq \lambda(f(\mathbf{c}^*) + \alpha^\alpha 2^{\alpha-1}\mathcal{E}_{\text{YDS}}(\boldsymbol{\tau}^*)) + \mu C(\boldsymbol{\tau}) \leq \alpha^\alpha 2^{\alpha-1}\lambda C^* + \mu C(\boldsymbol{\tau})\,.\end{aligned}$$

Similar to the proof of Theorem 3.5, we get PoA $\leq \alpha^{2\alpha} 2^{2\alpha-2}$. □

Next, we complement our results by proving a lower bound on the PoA that is not very far from our results by constructing an instance where the PoA is at least $\alpha^\alpha$. This instance is inspired by the instance used to prove the lower bound on the competitive ratio of AVR in [11]. We give the proof of the following theorem in the full version of the paper.

**Theorem 3.7.** *The online game induced by either IncrAVR$_{MW}$ or IncrAVR$_{SW}$ has PoA $\geq \alpha^\alpha$.*

# 4 Multiprocessor Case

In the following let AVR($m$) and OPT($m$) denote the multiprocessor AVR algorithm on $m$ processors (or its expended energy) as well as the corresponding optimal algorithm (and its expended energy). Both algorithms are formally defined and analyzed in [3].

In this section we define and analyze the PoA of a mechanism *Greedy(m)* induced by a simple greedy extension of AVR(1) to the multiprocessor setting. The mechanism induced by Greedy($m$) is an extension of $IncrAVR$ for both the single and the multi-window setting. In particular the strategy space and the volume of each job in each interval are the same, and therefore, also the charging with respect to the waiting cost. The only difference is that a greedy algorithm distributes this processing volume to the processors in each interval, which affects the energy consumption and the charging of the agents. More precisely, let $T = \cup_i \{t | t \text{ is start or endpoint of some interval in } I_i\}$, and let $t_1 < t_2 < \cdots < t_v$ be the distinct points in $T$. $T$ partitions the time-horizon into intervals $D_k := [t_k, t_{k+1})$. By definition, AVR(1) schedules exactly

$$w_i^k := s_i \cdot |D_k| = w_i \cdot |D_k|/|I_i|$$

amount of volume from $w_i$ in interval $D_k \subseteq I_i$.

**Algorithm Greedy(m):** Upon arrival of a job $i$, and for each
interval $D_k \subseteq I_i$ assign the volume $w_i^k$ to the processor $\ell$
that currently has the least load.

Note that Greedy($m$) is defined for both the single and the multi-window multiprocessor setting and that during execution of the algorithm $T$ may grow due to new job arrivals. Some interval $D_k$ may then get partitioned into subintervals. This does not change the assignment of jobs to processors: $w_i^k$ will just be split into two parts assigned to the newly created intervals, still on processor $\ell$.

We define the set of *Averaging Algorithms* for the problem, as those that assign the same amount of workload of each job to each interval as AVR(1). Both Greedy($m$) and AVR($m$) are averaging algorithms in the single window and the multi window setting.

**Definition 4.1.** *A scheduling algorithm A belongs to the class of* Averaging Algorithms*, if and only if for every job $i$ and interval $D_k \subseteq I_i$, A assigns workload $w_i^k := s_i \cdot |D_k|$ of $i$ to $D_k$. Any feasible schedule with this property is called an* Averaging Schedule*.*

In Subsection 4.2 we prove that Greedy($m$) achieves a constant competitive ratio with respect to energy. More formally, we prove the following theorem.

**Theorem 4.2.** *Greedy($m$) obtains a competitive ratio of $2^{3\alpha-1}\alpha^\alpha + 2^{2\alpha}$. Furthermore for any instance Greedy($m$) has an energy consumption that is upper bounded by $2^{2\alpha}$ times the energy consumption of the optimal averaging schedule.*

Aided by Theorem 4.2 we adapt the proofs of Section 3 in order to prove that the mechanism induced by Greedy($m$) attains a constant PoA. We note that the reason for attaining different PoA for the two settings is that in the single window setting YDS is not necessarily an averaging schedule and we lose another factor $2^{\alpha-1}\alpha^\alpha + 1$ compared to $YDS$. Otherwise the proofs are identical.

**Theorem 4.3.** *The online game induced by Greedy($m$) has PoA$\leq 2^{3\alpha-1}\alpha^\alpha$ in the multi-window setting.*

**Theorem 4.4.** *The online game induced by Greedy(m) has PoA* $\leq 2^{4\alpha-2}\alpha^{2\alpha}+2^{3\alpha-1}\alpha^{\alpha}$ *in the single window setting.*

## 4.1 PoA of the Induced Mechanism

Let $x_{j,k,\ell}$ be an indicator variable that is 1 if job $j$ is assigned to processor $\ell$ in interval $D_k$, and 0 otherwise. As before, we denote $s_j = \frac{w_j}{|I_j|}$ and $s_i^* = \frac{w_i^*}{|I_i^*|}$. Similar to Equation (5), in any strategy profile $\boldsymbol{\tau}$ agent $i$ is charged a payment of

$$\phi_i(\boldsymbol{\tau}) = \sum_{k:D_k \subseteq I_i} |D_k| \sum_{\ell=1}^{m} \left( \left( \sum_{j \leq i} s_j \cdot x_{j,k,\ell} \right)^{\alpha} - \left( \sum_{j<i} s_j \cdot x_{j,k,\ell} \right)^{\alpha} \right) .$$

Note that for each job $j$ and interval $D_k$ there is exactly one processor $\ell$ on which $j$ is executed and the indicator variable $x_{j,k,\ell} = 1$. We therefore can prove Lemma 3.3 for the multiprocessor setting as well, which we repeat here for the sake of completeness (the proof is deferred to the full version of the paper).

**Lemma 4.5.** *For any pair of strategy profiles* $\boldsymbol{\tau}$ *and* $\boldsymbol{\tau}^*$ *and for any* $\lambda, \mu$ *as in Equation* (3), $\sum_{i=1}^{n} C_i(\tau_i^*, \boldsymbol{\tau}_{-i}) \leq \lambda C(\boldsymbol{\tau}^*) + \mu C(\boldsymbol{\tau})$ .

*Proof sketch.* Since Lemma 3.4 is identical to the single processor case, it suffices to prove that $\sum_{i=1}^{n} \phi_i(\tau_i^*, \boldsymbol{\tau}_{-i}) \leq \lambda \phi(\boldsymbol{\tau}^*) + \mu \phi(\boldsymbol{\tau})$ holds. The analysis is essentially the same as in the single processor case, but slightly more involved since a job can be assigned to any of $m$ processors at each point in time. □

Theorems 4.3 and 4.4 can now be proven along the lines of Theorems 3.5 and 3.6, by taking into account the different competitive ratio of Greedy($m$). The formal proof is in the full paper again.

## 4.2 The Competitive Ratio of Greedy(m)

In this subsection we prove Theorem 4.2. The details regarding AVR($m$) can be found in [3]. We use the following lemma.

**Lemma 4.6** ([3])**.** *AVR(m) is* $(2^{\alpha-1}\alpha^{\alpha}+1)$*-competitive with respect to energy.*

We will make use of the following property, the proof of which is deferred to in the full version of the paper.

**Lemma 4.7.** *For any feasible schedule there exists an equivalent*[8] *schedule where in every interval* $D_k$*, every job* $i$ *is scheduled in at most two different processors.*

*Proof sketch.* Consider some feasible schedule $S$ where this is not the case for some interval $D_k$. Then one can rearrange the execution times of the jobs within $D_k$, "filling" the processors from left to right with the execution time of one job at a time. One can show that because $S$ is feasible, the total execution of a job will only split amongst two consecutive processors and that the total energy consumption or the fact that $S$ is an averaging schedule are not affected by this transformation. □

[8] Equivalent in the sense that the total energy consumption does not change and neither does the individual energy consumption or the resulting waiting cost. Furthermore if the initial schedule is an averaging one, then so is the resulting schedule.

The following lemma will be useful in our analysis.

**Lemma 4.8.** *For any input instance $L$ there exists an Averaging Schedule, in which each job appears on at most one processor in each interval $D_k$, with energy consumption no more than $2^\alpha$ times the energy consumed by the best (with respect to energy consumption) averaging schedule in $L$.*

*Proof sketch.* One can transform any schedule that does not satisfy the property into one that does, while increasing the energy consumption by at most a factor $2^\alpha$. W.l.o.g. we assume that each job runs at only one speed level within $D_k$, since if this is not the case we could process the job at the same timepoints on its average (for $D_k$) speed and improve the energy consumption. Consider a job $j$ that runs in two processors within $D_k$ (see Lemma 4.7). One can then "pack" the processing from one processor to the other one while at most doubling the speed. By repeating this for all jobs, each job appears on only one processor in $D_k$ and each job speed has at most doubled; the lemma follows. □

Lemma 4.8 together with Lemma 4.6 bounds the competitive ratio on the algorithm that always finds the best Averaging Schedule, with the extra restriction that each job can appear in at most one processor per interval. The reason why we would like to restrict Averaging Schedules further to only allow one processor per job and interval, is that it will help us keep the smoothness analysis simple. This additional constraint makes the problem computationally harder, yet we are able to reside to a simple, greedy algorithm which returns schedules that are close enough to the best Averaging Schedule without this constraint.

**Lemma 4.9.** *The energy consumption of Greedy($m$) is not more than $2^\alpha$ times the optimal energy consumption attainable by an Averaging Schedule which runs each job in at most one processor per interval, for any input instance.*

*Proof.* Consider some input instance $L$ and some interval $D_k$. Let $x_{i,k,\ell}$ be an indicator variable denoting whether job $i$ is assigned to processor $\ell$ in interval $D_k$. It is known [9] that Greedy($m$) is a 2-competitive algorithm with respect to the $L_p$-norm of loads assigned to the processors:

$$\left( \sum_{\ell=1}^{m} \left( \sum_{i:x_{i,k,\ell}=1} w_i^k \right)^\alpha \right)^{\frac{1}{\alpha}}.$$

The lemma directly follows by the definition of the competitive ratio and raising to the power $\alpha$. □

So in total, for any instance $L$, Greedy($m$) returns an Averaging Schedule with an energy consumption within a factor $2^\alpha$ of that of the optimal Averaging Schedule for $L$ that is constrained to at most one processor per job per interval (Lemma 4.9). In turn the optimal Averaging Schedule for $L$ constrained to at most one processor per job per interval is within another factor $2^\alpha$ from the optimal averaging schedule (and therefore also a factor $2^\alpha$ from AVR($m$)) (Lemma 4.8). Finally, AVR($m$) is within a $(2^{\alpha-1}\alpha^\alpha + 1)$-factor from the optimal energy consumption attainable for $L$ (Lemma 4.6).

**Remark 4.10.** We note that a better PoA can be obtained by partitioning the jobs into pieces in an intricate way, so that considering these pieces as independent jobs and releasing them in a specific order, would result in AVR($m$) producing a feasible schedule for the original job set, and furthermore, each piece being assigned to exactly one processor. This results in a slightly better PoA, but we defer this more involved derivation to the full version of the paper.

# 5 Bayesian Setting

In this section we study the Bayesian setting. In order to keep the presentation simple, we show the PoA upper bound for the single processor case and give the result of the multiprocessor case as a corollary at the end. We first show that the Bayesian game (induced be either $\text{IncrAVR}_{\text{MW}}$ or $\text{IncrAVR}_{\text{SW}}$) also satisfies the smoothness property, and then by using the *Extension Theorem* [31] we derive the same PoA bounds for this more general setting. We remark that the total cost is a cost-dominated minimization objective, meaning that it is always at most as large as the sum of agents' costs. In particular, $C(\mathbf{t};\boldsymbol{\tau}) = \sum_i C_i(t_i;\boldsymbol{\tau})$ and we will use this fact in the Extension Theorem. We first restate the definition of *Smooth Game*.

**Definition 5.1.** (Smooth Game [31])
*A cost-minimization game is $(\lambda,\mu)$-smooth with respect to a strategy profile $\boldsymbol{\tau}^* : \mathbf{T} \to \mathbf{A}$ if for all type vectors $\mathbf{t},\mathbf{q} \in \mathbf{T}$ and any strategy profile $\boldsymbol{\tau}$ feasible for $\mathbf{q}$,*

$$\sum_{i=1}^{n} C_i(t_i;(\tau_i^*(\mathbf{t}),\boldsymbol{\tau}_{-i})) \leq \lambda C(\mathbf{t};\boldsymbol{\tau}^*(\mathbf{t})) + \mu C(\mathbf{q};\boldsymbol{\tau})\,. \tag{6}$$

Next, we show that the game induced by the IncrAVR mechanism is $(\lambda,\mu)$-smooth for the $\lambda$ and $\mu$ given by Lemma 3.2.

**Lemma 5.2.** *$IncrAVR_{MW}$ and $IncrAVR_{SW}$ are $(\lambda,\mu)$-smooth for the $\lambda$ and $\mu$ given in Equation (3).*

The proof of Lemma 5.2, in the full version, follows the proof of Lemma 3.3 by carefully considering the agents' types. Finally, by using the Extension Theorem [31] and the values of $\lambda$ and $\mu$ of Theorem 3.5, we derive the same bounds as in the online case.

**Theorem 5.3.** (Extension Theorem [31])
*If a game is $(\lambda,\mu)$-smooth with respect to an optimal choice function for a cost-dominated minimization objective $C$, then the PoA of the Bayesian setting with respect to $C$ is at most $\lambda/(1-\mu)$.*

The Extension Theorem gives a bound of $\lambda/(1-\mu)$ with respect to the cost function $C$. In the case of $\text{IncrAVR}_{\text{MW}}$, minimization of $C$ results in the social optimum. However, for $\text{IncrAVR}_{\text{SW}}$ this is not the case and similarly to the proof of Theorem 3.6 we need to multiply the PoA bound by an extra factor of $\alpha^\alpha 2^{\alpha-1}$.

**Corollary 5.4.** *The Bayesian game induced by $IncrAVR_{MW}$ has PoA $\leq \alpha^\alpha 2^{\alpha-1}$.*

**Corollary 5.5.** *The Bayesian game induced by $IncrAVR_{SW}$ has PoA $\leq \alpha^{2\alpha}2^{2\alpha-2}$.*

By applying the analysis of the Bayesian setting to the multiprocessor case we can immediately derive the following corollaries.

**Corollary 5.6.** *The Bayesian game induced by Greedy($m$) with multi-window strategies in the multiprocessor case has PoA $\leq 2^{3\alpha-1}\alpha^\alpha$.*

**Corollary 5.7.** *The Bayesian game induced by Greedy($m$) in with single window strategies the multiprocessor case has PoA $\leq 2^{4\alpha-2}\alpha^{2\alpha} + 2^{3\alpha-1}\alpha^\alpha$.*

## References


[1] Susanne Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, 2010.

[2] Susanne Albers and Antonios Antoniadis. Race to idle: New algorithms for speed scaling with a sleep state. *ACM Trans. Algorithms*, 10(2):9:1–9:31, 2014.

[3] Susanne Albers, Antonios Antoniadis, and Gero Greiner. On multi-processor speed scaling with migration. *J. Comput. Syst. Sci.*, 81(7):1194–1209, 2015.

[4] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Trans. Algorithms*, 3(4):49, 2007.

[5] Anders Andrae and Tomas Edler. On global electricity usage of communication technology: trends to 2030. *Challenges*, 6(1):117–157, 2015.

[6] Eric Angel, Evripidis Bampis, Vincent Chau, and Nguyen Kim Thang. Throughput maximization in multiprocessor speed-scaling. In *Algorithms and Computation - 25th International Symposium, ISAAC 2014*, volume 8889 of *Lecture Notes in Computer Science*, pages 247–258. Springer, 2014.

[7] Antonios Antoniadis and Andrés Cristi. A near optimal mechanism for energy aware scheduling. In *SAGT*, 2018.

[8] Antonios Antoniadis, Chien-Chung Huang, and Sebastian Ott. A fully polynomial-time approximation scheme for speed scaling with sleep state. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 1102–1113. SIAM, 2015.

[9] Adi Avidor, Yossi Azar, and Jirí Sgall. Ancient and new algorithms for load balancing in the $l_{\mathrm{p}}$ norm. *Algorithmica*, 29(3):422–441, 2001.

[10] Moshe Babaioff, Yishay Mansour, Noam Nisan, Gali Noti, Carlo Curino, Nar Ganapathy, Ishai Menache, Omer Reingold, Moshe Tennenholtz, and Erez Timnat. Era: A framework for economic resource allocation for the cloud. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 635–642. International World Wide Web Conferences Steering Committee, 2017.

[11] Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. Average rate speed scaling. *Algorithmica*, 60(4):877–889, 2011.

[12] Nikhil Bansal, Ho-Leung Chan, Dmitriy Katz, and Kirk Pruhs. Improved bounds for speed scaling in devices obeying the cube-root rule. *Theory of Computing*, 8(1):209–229, 2012.

[13] Nikhil Bansal, Ho-Leung Chan, Tak Wah Lam, and Lap-Kei Lee. Scheduling for speed bounded processors. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 409–420. Springer, 2008.

[14] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1):3:1–3:39, 2007.

[15] Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM J. Comput.*, 39(4):1294–1308, 2009.

[16] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Y. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, 82:47–111, 2011.

[17] David M Brooks, Pradip Bose, Stanley E Schuster, Hans Jacobson, Prabhakar N Kudva, Alper Buyuktosunoglu, John Wellman, Victor Zyuban, Manish Gupta, and Peter W Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[18] Ho-Leung Chan, Jeff Edmonds, Tak Wah Lam, Lap-Kei Lee, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Nonclairvoyant speed scaling for flow and energy. *Algorithmica*, 61(3):507–517, 2011.

[19] Shuchi Chawla, Nikhil Devanur, Janardhan Kulkarni, and Rad Niazadeh. Truth and regret in online scheduling. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, pages 423–440. ACM, 2017.

[20] Rajarshi Das, Jeffrey O. Kephart, Charles Lefurgy, Gerald Tesauro, David W. Levine, and Hoi Chan. Autonomic multi-agent management of power and performance in data centers. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, 2008, Industry and Applications Track Proceedings*, pages 107–114, 2008.

[21] M. Dayarathna, Y. Wen, and R. Fan. Data center energy consumption modeling: A survey. *IEEE Communications Surveys Tutorials*, 18(1):732–794, Firstquarter 2016.

[22] Christoph Dürr, Lukasz Jez, and Oscar C. Vásquez. Scheduling under dynamic speed-scaling for minimizing weighted completion time and energy consumption. *Discrete Applied Mathematics*, 196:20–27, 2015.

[23] Christoph Dürr, Lukasz Jez, and Oscar C. Vásquez. Mechanism design for aggregating energy consumption and quality of service in speed scaling scheduling. *Theor. Comput. Sci.*, 695:28–41, 2017.

[24] Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

[25] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Conference on Theoretical Aspects of Computer Science*, STACS'99, pages 404–413, Berlin, Heidelberg, 1999. Springer-Verlag.

[26] Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Online speed scaling based on active job count to minimize flow plus energy. *Algorithmica*, 65(3):605–633, 2013.

[27] Brendan Lucier, Ishai Menache, Joseph Seffi Naor, and Jonathan Yaniv. Efficient online scheduling for deadline-sensitive jobs. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, pages 305–314. ACM, 2013.

[28] Nicole Megow and José Verschae. Dual techniques for scheduling on a machine with varying speed. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 745–756. Springer, 2013.

[29] Hervé Moulin. Incremental cost sharing: Characterization by coalition strategy-proofness. *Social Choice and Welfare*, 16(2):279–320, 1999.

[30] Trevor Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.

[31] Tim Roughgarden. The price of anarchy in games of incomplete information. *ACM Trans. Econ. Comput.*, 3(1):6:1–6:20, March 2015.

[32] Bolei Xu, Tao Qin, Guoping Qiu, and Tie-Yan Liu. Competitive pricing for cloud computing in an evolutionary market. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, pages 1755–1756, 2015.

[33] F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced CPU energy. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 374–382. IEEE Computer Society, 1995.