Proceedings of the of the 17th Conference on Computer Science and Intelligence Systems pp. 373–382
DOI: 10.15439/2022F52
ISSN 2300-5963 ACSIS, Vol. 30

Stackelberg Strategies for Weighted Load Balancing Games

Neta Stein and Tami Tamir
School of Computer Science
Reichman University (IDC)
Herzliya, Israel
Emails: neta.stein@post.idc.ac.il , tami@idc.ac.il

***Abstract*—An instance of a weighted Stackelberg load balancing game is given by a set of identical machines, a set of variable-length jobs and a parameter $0 \leq \alpha \leq 1$. A centralized authority, denoted *the leader*, selects a subset of the jobs whose total length is at most an α-fraction of the total length and determines their assignment on the machines. After the controlled jobs are assigned, the remaining jobs join the schedule. They act selfishly, each determining its own assignment.**

Our work combines theoretical and experimental results for this setting. We suggest various heuristics for the leader and analyze their performance.

I. Introduction

IN RESOURCE allocation applications, a set of resources is used by a set of clients (users). For example, in job-scheduling applications, servers process jobs; in communication or transportation networks, traffic is routed on network links. In some settings, the users are assigned to the different resources by a centralized authority. The centralized authority is aware of all clients' requests and can utilize the system in the best possible way. Other resource allocation applications lack a central authority and are managed by multiple strategic users, whose individual payoff is affected by the assignment of other users. As a result, game theory has become an essential tool in the analysis of resource-allocation services. In the corresponding game, every client is a selfish player who aims at maximizing his own utilization.

In this work, we consider systems in which the two models are combined. That is, some of the users obey a central authority and some are selfish. The goal of the system is to assign the centrally controlled users such that the total system performance is optimized. We formulate this goal as an optimization problem via *Stackelberg games*, in which one player, corresponding to the centralized authority, acts as *a leader* and the rest of the players, corresponding to the selfish users, as *followers*. The problem is then to compute a strategy for the leader (a Stackelberg strategy) that cause the followers to react in a way that is as good as possible for the social utility.

We focus on load balancing games in job scheduling systems. The users are jobs having variable weights and the resources are machines. Every job should be assigned on a single machine. The cost of a job depends on the total load on its machine and the social cost is given by the load on the most loaded machine. In centralized systems, this load balancing scenario is the well-studied *minimum Makespan* problem. It was well-studied also as a game where all the jobs are selfish, however, it has not been studied in an environment with a mixed type of jobs. All previous works on Stackelberg strategies consider non-atomic (splittable) players – every player has a neglected load and the total load can be distributed among the machines in an arbitrary way. Thus, our work initiate the study of Stackelberg strategies for weighted singleton congestion games.

Our goal is to develop Stackelberg strategies that guarantee load balancing and a low social cost. We will consider several different models, depending on the different capabilities of the leader.

In our model, the leader can select the jobs it controls and its power is given by a parameter $0 \leq \alpha \leq 1$, such that it can select any subset of jobs whose total load is at most an α fraction of the total load.

As we show, in some instances, the leader should better exploit its power only partially and let more jobs select their strategy selfishly. Intuitively, this is due to the fact that the leader acts first and its assignment is irrevocable.

The scenarios we propose to study arise in real life applications that provide service to a mixture of independent and controlled clients. For example, several companies suggest computing services on shared machines for private users (using cloud services) as well as local computation tasks. Similarly, in routing problems, some users obey and consult a navigation app, while others don't. The navigation app acts as a leader that determines the decisions (selected path) of some of the clients.

A. Notation and Problem Statement

A *load balancing game* is given by $G = \langle \mathcal{J}, \mathcal{M}, \{p_j\}\ \forall j \in \mathcal{J} \rangle$, where $\mathcal{J}$ is a set of n *jobs*, $\mathcal{M}$ is a set of m *machines*, p_j is the *weight* (also denoted *size*) of job j. In unweighted games, all the jobs have the same unit size, that is, $\forall j, p_j = 1$.

A *Stackelberg load balancing game* is given by $\langle G, \alpha \rangle$, where G is a load balancing game and $0 \leq \alpha \leq 1$ denotes the maximal fraction of the load controlled by the leader.

Let A be the set of jobs controlled by the leader. Let $P = \sum_j p_j$. We have that $\sum_{j \in A} p_j \leq \alpha P$. A profile of a game is a *schedule* $s = \langle s_1, \ldots, s_n \rangle \in M^n$ describing the machines on

which the jobs are assigned[1]. If $j \in A$ then s_j is determined by the leader and if $j \in \mathcal{J} \setminus A$, then s_j is selected by player j.

For a machine $i \in \mathcal{M}$, the *load* on i in s, denoted $L_i(s)$, is the total size of the jobs assigned on machine i in s, that is, $L_i(s) = \sum_{\{j|s_j=i\}} p_j$. When s is clear from the context, we omit it. It takes p_j time-units to process job j on machine i. As common in the study of job-scheduling games, we assume that all the jobs assigned on the same machine are processed in parallel and have the same cost, defined as the machine's completion time. Formally, the cost of job j in profile s is $C_j(s) = L_{s_j}(s)$. The players that control jobs act selfishly, trying to minimize the cost of the job they control. The leader is trying to optimize some global objective function. Note that if $A = \mathcal{J}$ then all the jobs are controlled by the leader and we have a classical job scheduling problem.

The leader applies its strategy first and determines an assignment for the players in A. Afterwards, the players in $\mathcal{J} \setminus A$ join this partial schedule, each selecting selfishly an assignment for its job. The jobs assigned by the leader cannot change their assignment. The selfish jobs may change their assignment in response to other jobs' assignments.

For a profile s, a job $j \in \mathcal{J} \setminus A$ and a machine $s'_j \neq s_j$, let (s_{-j}, s'_j) denote the profile obtained from s by replacing the strategy of job j by s'_j. That is, the profile resulting from a migration of job j from machine s_j to machine s'_j. A profile s is a *pure Nash equilibrium* (NE) if no selfish job can benefit from unilaterally deviating from its strategy in s to another strategy; i.e., for every job $j \in \mathcal{J} \setminus A$ and every machine s'_j it holds that $C_j(s_{-j}, s'_j) \geq C_j(s)$. The cost of a schedule is defined to be the maximal completion time of a job, also known as *the Makespan* of the schedule, that is, $cost(s) = \max_{j \in \mathcal{J}} C_j(s)$.

It is well known that decentralized decision-making may lead to sub-optimal solutions from the point of view of society as a whole. For a game G, let $S(G)$ be the set of feasible profiles of G. We denote by $OPT(G)$ the cost of a social optimal (SO) solution; i.e., $OPT = \min_{s \in S(G)} cost(s)$. We quantify the inefficiency incurred due to self-interested behavior according to the *price of anarchy* (PoA) [16] and *price of stability* (PoS) [1], [20] measures. The PoA is the worst-case inefficiency of a pure Nash equilibrium, while the PoS measures the best-case inefficiency of a pure Nash equilibrium. Formally,

Definition 1.1: Let $\mathcal{G}$ be a family of games and let G be a game in $\mathcal{G}$. Let $\Upsilon(G)$ be the set of pure Nash equilibria of the game G. Assume that $\Upsilon(G) \neq \emptyset$.

- The *price of anarchy* of G is the ratio between the *maximal* cost of a NE and the social optimum of G. That is, $\text{PoA}(G) = \max_{s \in \Upsilon(G)} cost(s)/OPT(G)$. The *price of anarchy* of the family of games $\mathcal{G}$ is $\text{PoA}(\mathcal{G}) = sup_{G \in \mathcal{G}} \text{PoA}(G)$.

We now define the inefficiency measure of a Stackelberg game.

[1]In this paper, we only consider *pure* strategies. Unlike mixed strategies, pure strategies may not be random or drawn from a distribution.

Definition 1.2: Let $\langle \mathcal{G}, \alpha \rangle$ be a family of Stackelberg games where $\mathcal{G}$ is a family of games and let G be a game in $\mathcal{G}$. Let $\Upsilon(G, \alpha)$ be the set of pure Nash equilibria of the game $\langle G, \alpha \rangle$. Assume that $\Upsilon(G, \alpha) \neq \emptyset$.

- The *price of anarchy* of $\langle G, \alpha \rangle$ is the ratio between the *maximal* cost of a NE and the social optimum of G. That is, $\text{PoAL}(G, \alpha) = \max_{s \in \Upsilon(G,\alpha)} cost(s)/OPT(G)$. The *price of anarchy* of the family of Stackelberg games $\langle \mathcal{G}, \alpha \rangle$ is $\text{PoAL}(\mathcal{G}, \alpha) = sup_{G \in \mathcal{G}} \text{PoAL}(G, \alpha)$.

B. Related Work

The two extreme cases, of $A = \mathcal{J}$ and $A = \emptyset$ are well studied. The case $A = \mathcal{J}$ is the classical minimum makespan problem, while the case $A = \emptyset$ is the classical load balancing game.

The minimum makespan problem is an NP-hard problem and has a rich collection of approximation algorithms. For identical machines, the simple greedy List-scheduling (LS) algorithm [10] provides a $(2 - \frac{1}{m})$-approximation to this problem. A bit better approximation ratio of $(\frac{4}{3} - \frac{1}{3m})$ is guaranteed by the Longest Processing Time (LPT) algorithm [11]. A PTAS for the minimum makespan problem on identical machines is given in [12]. For related machines (with various speeds), List-scheduling guarantees $\Theta(m)$-approximation [4] and a PTAS is presented in [6].

Load balancing *game* consist of a set of jobs (players) and a set of machines. Each job is controlled by a selfish agent who aims to minimize his cost - given by the load on the machine it is assigned to ([21]). The concept of the price of anarchy (PoA) was introduced by Koutsoupias and Papadimitriou in [16]. They proved that the price of anarchy of job scheduling games is $2 - \frac{1}{m}$. In [7], Finn and Horowitz presented an upper bound of $2 - \frac{2}{m+1}$ for the price of anarchy in load balancing games with identical machines. Czumaj and Vöcking [5] gave tight bounds for related machines that grow as the number of machines grows.

Other related work consider Stackelberg strategies for resource allocation games with identical players and non-decreasing latency functions which can vary between the resources. In these games, the leader is characterized by the fraction $\alpha \in [0, 1]$ of the players it controls and an optimal allocation can be computed in polynomial time. Our model differs from the previous studies as we do not assume the players to have identical weight or strategy space, hence, the leader is characterized by the specific set of players it controls and calculating a socially optimum assignment is an NP-hard problem.

For routing games on parallel networks for with non-decreasing latency functions, it is known that computing an optimal Stackelberg strategy is NP-hard. There are two known approximation algorithms for the case of splittable (non-atomic) symmetric games with non-decreasing latency functions presented by Roughgarden in [18]; The *Scale* strategy which simply employs the optimal configuration scaled by the fraction of coordinated players . The best bound known for *Scale* PoA is $\frac{4}{3} - \frac{X}{3}$ where $X = \frac{(1-\sqrt{1-\alpha})(3\sqrt{1-\alpha}+1)}{2\sqrt{1-\alpha}+1}$

[13]. The *LLF* (Largest Latency First) strategy presented by Roughgarden in [18] assigns the controlled players to the largest cost strategies in the optimal configuration. The best known upper bound, achieved by Karakostas and Kolliopoulos [13], is equal to $\frac{4}{3}$ for $\alpha \leq \frac{1}{3}$ and $\frac{2(1-\alpha^2)}{2-\alpha-\sqrt{4\alpha-3\alpha^2}}$ for $\alpha > \frac{1}{3}$.

Subsequently, Kumar and Marathe [17] presented a fully polynomial-time approximation scheme for the problem of computing an optimal Stackelberg configuration on parallel links with polynomial latencies. Stackelberg routing in arbitrary networks is studied in [2].

The work mostly related to our study considers games with unsplittable (atomic) flows with non-decreasing latency functions. In these games, different machines may have different latency functions, that are not necessarily linear. The players are identical, but they do have a non-splittable constant size. Thus, the problem of calculating the optimal assignment is polynomially solvable, contrary to our model. Fotakis [8] studied *LLF* and a randomized version of *Scale* and gave upper and lower bounds for them. He also introduced the Stackelberg strategy λ-*Cover* which assigns to every resource either at least λ or as many coordinated players as the resource has in the social optimum. Finally, he also gave upper bounds for strategies obtained by combining λ-Cover with either *LLF* or *Scale* and upper bounds for games played on parallel links.

Vittorio and Vinci [3] then presented improved bounds for the three Stackelberg strategies studied by Fotakis [8]. For *LLF* they showed *PoA* of exactly $\frac{20-11\alpha}{8}$ for $\alpha \in [0, \frac{4}{7}]$ and $\frac{4-3\alpha+\sqrt{4\alpha-3\alpha^2}}{2}$ for $\alpha \in [\frac{4}{7}, 1]$. For λ-*Cover* they showed that the *PoA* is for affine functions is $\frac{4\lambda-1}{3\lambda-1}$ and $1 + \frac{4\lambda+1}{4\lambda(2\lambda+1)}$ for linear ones. Finally, for *Scale* they give a bound of $1 + \frac{(1-\alpha)(2h+1)}{(1-\alpha)h^2+\alpha h+1}$, where h is the unique positive integer such that $\alpha \in [r_L(h), r_U(h)]$, with $r_L(h) = \frac{2h^2-3}{2(h^2-1)}$, $r_U(h) = \frac{2h^2+4h-1}{2h(h+2)}$ and $r_L(1) = 0$.

To the best of our knowledge our work is the first to consider a model with unsplittable variable size jobs.

C. Our Results

Let $\langle G, \alpha \rangle$ be a Stackelberg load balancing game. The leader aims to minimize the makespan of the schedule achieved after the addition of selfish jobs. We consider two questions:

1) How should the leader choose the jobs it controls?
2) How should it schedule them on the machines?

In Section II, we present results for games with two job sizes, that is, for all $j \in \mathcal{J}$, $p_j \in \{1, p\}$. First, in Section II-A we characterize the social optimum in these instances in leader-free settings. In Section II-B, we consider settings where the leader must exploit all of its power and we show that it may result with a worst schedule than a NE in a leader-free environment. In Section II-C we show that when $\alpha \geq \min\{\frac{n_1}{P}, \frac{n_p p}{P}\}$ the leader can guarantee an optimal NE. Furthermore, we show that for every $\alpha \geq 0$, for every game G with two job sizes, $PoAL(G, \alpha) \leq 1 + \frac{p-1}{OPT(G)}$.

In Section III, we analyze the PoAL as a function of α and compare it to the PoA in a leader-free game. We show that for $\alpha < \frac{1}{m+1}$, $PoAL(\mathcal{G}, \alpha) = PoA(\mathcal{G})$. We also show a lower bound of $\alpha \geq \frac{1}{m}$ from which we can guarantee $PoAL(\mathcal{G}, \alpha) < PoA(\mathcal{G})$. Furthermore, we present and analyze two strategies for choosing and scheduling the controlled jobs.

In Section IV, we present the leader's heuristics for both choosing the controlled jobs and assigning them to machines. In Section V we present our experimental results.

We conclude in Section VI, where we summarize the work and suggest some directions for future work. Due to space constraints, some of the technical proofs are omitted.

II. Instances with Two Job Sizes $\{1, p\}$

In this section, we consider the class $\mathcal{G}_2$ of game instances with only two different job sizes, w.l.o.g., 1 and p. Formally, $G \in \mathcal{G}_2$ if there exists a constant $p > 1$ such that $\forall j \in \mathcal{J}$, $p_j \in \{1, p\}$. We denote by ℓ-job a job of size ℓ. Let n_1 and n_p denote the number of 1-jobs and p-jobs, respectively. It is easy to see that LPT algorithm is optimal for any $G \in \mathcal{G}_2$.

A. Social Optimum in Leader-Free Games

We first provide some simple observations regarding the social optimum and leader-free games in $\mathcal{G}_2$.

First, we denote r_G to be the number of machines with exactly $\lceil \frac{n_p}{m} \rceil$ p-jobs on them on a LPT schedule. Thus, $r_G = n_p \mod m$ when $\frac{n_p}{m} \notin \mathbb{N}$ and there are $m - r_G$ machines with $\lfloor \frac{n_p}{m} \rfloor$ p-jobs. Also, let $h = \lfloor \frac{n_p}{m} \rfloor$, meaning, $n_p = hm + r_G$.

Proposition 2.1: For every load balancing game $G \in \mathcal{G}_2$,

$$OPT(G) = \begin{cases} \lceil \frac{n_p}{m} \rceil p & \text{if } n_1 \leq (m - r_G)p \text{ and } n_p > 0 \\ \lceil \frac{P}{m} \rceil & \text{otherwise} \end{cases}$$

Proof: Consider an optimal LPT schedule of $G \in \mathcal{G}_2$. If $n_p > 0$ then LPT first schedules all the p-jobs in a balanced way. The makespan after this stage is $\lceil \frac{n_p}{m} \rceil p$. For the makespan to remain $\lceil \frac{n_p}{m} \rceil p$, the number of 1-jobs must be at most $(m - r_G)p$, since otherwise, $\frac{n_1 + n_p p}{m} > \lceil \frac{n_p}{m} \rceil p$. If $n_1 > (m - r_G)p$, then the number of 1-jobs is sufficient to perfectly balance the load on the machines, up to a gap of 1, hence, the resulting schedule has a makespan of $\lceil \frac{P}{m} \rceil$. ∎

Proposition 2.2: For every load balancing game $G \in \mathcal{G}_2$, if $PoA(G) > 1$, then in every sub-optimal NE s, every machine a with $L_a(s) > OPT(G)$ processes only p-jobs.

Proof: Let $G \in \mathcal{G}_2$ such that $PoA(G) > 1$. Let s be a NE of G such that $cost(s) > OPT(G)$. Let a be a machine with $L_a(s) > OPT(G)$. By the pigeonhole principle, there must be a machine b for which $L_b < OPT(G)$. Hence, $L_a(s) > L_b(s) + 1$. Machine a processes only p-jobs, as otherwise, any 1-job on machine a would benefit from migrating to machine b, contradicting the stability of s. ∎

Next, we characterize instances for which every NE is optimal. First, we calculate some useful values. Denote $G \in \mathcal{G}_2$ to be a game with $n_1 > (m - r_G)p$. In order to calculate the value of $OPT(G)$ we take s to be an LPT schedule which we know is optimal. Each machine has at least h p-jobs on it and there are $m - r_G$ machines with exactly h which has exactly $(m - r_G)p$ 1-jobs scheduled on them since $OPT(G) > \lceil \frac{n_p}{m} \rceil$. Thus, we have n_p p-jobs and $(m - r_G)p$ 1-jobs in order to have all the machines with load of $\lceil \frac{n_p}{m} \rceil$. Hence, there are

more $n_1-(m-r_G)p$ 1-jobs in $\mathcal{J}$ and the maximal load they reach is $\lceil\frac{n_1-(m-r_G)p}{m}\rceil$. Denote this value to be B_G and we get in total $OPT(G)=\lceil\frac{P}{m}\rceil=\lceil\frac{n_p}{m}\rceil+B_G$.

For some schedule s' if there is a machine i with $L_i(s')>OPT(G)$ then it must be that there are only p-jobs on i. The number of p-jobs on i is $\lceil\frac{n_p}{m}\rceil+\lfloor\frac{B_G}{p}\rfloor+1$, meaning, the number of p-jobs that fits in a machine with load $OPT(G)$ and additional 1 p-job. We denote the number of p-jobs on i to be C_G. Using the above definitions, we can characterize instances for which every NE is optimal (proof omitted).

Proposition 2.3: For every load balancing game $G\in\mathcal{G}_2$, $PoA(G)=1$ iff at least one of the following conditions holds:

1) $n_p\leq 1$
2) $OPT(G)=\lceil\frac{n_p}{m}\rceil p$
3) $OPT(G)>\lceil\frac{n_p}{m}\rceil p$ and $n_1<(-(n_p-C_G)\mod m-1)p$ or $C_Gp>\lfloor\frac{(n_p-C_G)p+n_1}{m-1}\rfloor+p$ or $(n_pp-1)(m-1)\leq n_1$

B. The Leader Controls Maximal Load from αP

In this section we assume that the leader must fully exploit its power, that is, the leader must control a subset of jobs whose total size is maximal among subsets of total size at most αP. We assume that the leader has the computational power to identify such a subset. In particular, if there exists a subset of the jobs with total size αP, then the leader must control such a subset. A game in which the leader must control such a subset is denoted $\langle G,_{=}\alpha\rangle$.

The definition of PoAL is adjusted to fit this case as follows:

Definition 2.1: Let $\langle\mathcal{G},_{=}\alpha\rangle$ be a family of Stackelberg games, and let $G\in\mathcal{G}$. Let $\Upsilon(G,_{=}\alpha)$ be the set of pure Nash equilibria of the game $\langle G,_{=}\alpha\rangle$. Assume that $\Upsilon(G,_{=}\alpha)\neq\emptyset$.

- The *price of anarchy* of $\langle G,_{=}\alpha\rangle$ is the ratio between the *maximal* cost of a NE and the social optimum of G. That is, $\text{PoAL}(G,_{=}\alpha)=\max_{s\in\Upsilon(G,_{=}\alpha)}cost(s)/OPT(G)$. The *price of anarchy* of the family of Stackelberg games $\langle\mathcal{G},_{=}\alpha\rangle$ is $\text{PoAL}(\mathcal{G},_{=}\alpha)=sup_{G\in\mathcal{G}}\text{PoAL}(G,_{=}\alpha)$.

We first show that controlling the maximal possible load may not always minimize the PoAL.

Proposition 2.4: There exists a game $G\in\mathcal{G}_2$ and $0\leq\alpha\leq 1$ such that $PoAL(G,_{=}\alpha)>PoA(G)$.

Proof: Let G be a weighted Stackelberg load balancing game with $m=2$, $p=4$, $n_p=3$ and $n_1=2$ and let $\alpha=\frac{5}{14}$. Let $\mathcal{J}=\{j_1\dots j_5\}$ where $p_1=p_2=p_3=4$ and $p_4=p_5=1$.

We have $P=14$, thus, the leader must control jobs of total size 5. This implies that the leader controls one 4-job and one 1-job. W.l.o.g. the leader controls jobs j_1 and j_4. Figure $1(a)$ presents an optimal schedule of $\mathcal{J}$. $OPT(G)=8$ and by Proposition 2.1 $PoA(G)=1$. We show that $PoAL(G,_{=}\alpha)>1$. If the leader schedules both of the jobs on the same machine, w.l.o.g., on machine m_1, then a possible NE is depicted in Figure $1(b)$. We have $L_1(s)=9$ and $L_2(s)=5$. The only job that may benefit from a migration is the 1-job on m_1. However, this job was assigned by the leader and cannot change its assignment.

If the leader schedules each of the two controlled jobs on a different machine, w.l.o.g., $s_1=1$ and $s_4=2$, then a possible NE is a one where j_2,j_3 are scheduled on m_1 and j_5 on m_2. We have, $L_1(s)=9$ and $L_2(s)=5$ as depicted in Figure $1(c)$. The only job that may benefit from a migration is the 1-job on m_1. However, this job was assigned by the leader and cannot change its assignment.

Hence, $PoAL(G)=\frac{9}{8}>1=PoA(G)$. ∎

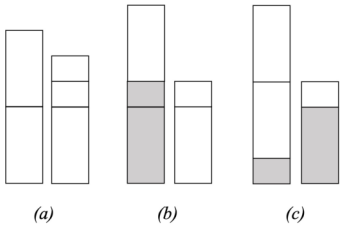

Fig. 1. $PoAL(G)>PoA(G)$. (a) optimal and worst case schedule with only selfish jobs, (b),(c) possible NE profiles with controlled jobs

We show that there is no constant $\alpha<1$ such that controlling α of the total load is sufficient to guarantee an optimal assignment.

Theorem 2.5: For every $\epsilon>0$ and every $p>1$, there exists a load balancing game $G\in\mathcal{G}_2$ such that $PoAL(G,_{=}\alpha)\geq 1+\frac{1}{p}$ for $\alpha\geq 1-\epsilon$.

Proof: Given ϵ,p, let $m\geq 3$ be an integer such that $\frac{2p-1}{m}\leq\epsilon$ and let $\epsilon'=\frac{2p-1}{pm}$. Let G be a game on m machines. Let $n_1=p$ and $n_p=m-1$. Thus, $P=pm$. Since the leader controls a fraction $1-\epsilon'$ of the total load and $\epsilon'P=2p-1$, it must be that the leader does not control exactly one p-job and $p-1$ 1-jobs, that is, A consists of a single 1-job and $m-2$ p-jobs.

In every optimal schedule for G, all the machines have load p. Specifically, on $m-1$ machines there is one p-jobs and on one machines there are p 1-jobs. Thus, $OPT(G)=\lceil\frac{P}{m}\rceil=p$.

In order to achieve an optimal solution, the leader must schedule the jobs in A in a sub-assignment of an optimal schedule, since otherwise it cannot be completed to an optimal schedule. We show that the leader cannot schedule the jobs of A such that any stable addition of the selfish jobs is optimal.

The only possible assignment for the leader, is to schedule $m-2$ p-jobs on $m-2$ machines and the single 1-job on another machine. After the assignment of A, there is an empty machine. A possible stable assignment of the selfish jobs is produced by adding one p-job to the machine with load 1 and assigning the $p-1$ 1-jobs on the empty machine. (see Figure 2)

In this case, it is easy to see that the resulting schedule is stable against deviations of the selfish jobs, thus, the resulting schedule has cost $p+1$ and $PoAL=1+\frac{1}{p}$. ∎

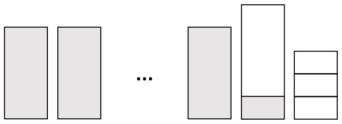

Fig. 2. The only optimal Stackelberg strategy of the controlled (shaded) jobs and a possible non-optimal stable completion of the assignment by the selfish (light) jobs.

C. Games with Optional Control

As shown in section II-B, forcing the leader to fully exploit its power may be harmful. We therefore consider next the more flexible and reasonable setting in which the leader may choose the amount of load it controls out of the maximal allowed fraction α.

Theorem 2.6: For every game $\langle G, \alpha \rangle$ with $G \in \mathcal{G}_2$ and $\alpha \geq \min\{\frac{n_1}{P}, \frac{n_p p}{P}\}$ it holds that $PoAL(G, \alpha) = 1$.

Proof: Let $G \in \mathcal{G}_2$. If $\alpha \geq \frac{n_p p}{P}$, the leader may choose to control all of the p-jobs and none of the 1 jobs. If $\alpha \geq \frac{n_1}{P}$, it can choose to control all of the 1-jobs. The leader would schedule the controlled jobs in a sub-assignment of an optimal schedule. Since all of the selfish jobs have the same size, they will complete the assignment in a balanced schedule, which is optimal. ∎

Theorem 2.7: For every $\alpha \geq 0$ and $G \in \mathcal{G}_2$, $PoAL(G, \alpha) \leq 1 + \frac{p-1}{OPT(G)}$.

Proof: Let $G \in \mathcal{G}_2$ be a weighted Stackelberg load balancing game. Assume that the leader assigns the controlled jobs in a sub-assignment of an optimal schedule. We show that for any completion of this sub-assignment to a NE, s, it holds that $cost(s) \leq OPT(G) + p - 1$.

Assume by contradiction that there is a machine m_i with $L_i(s) \geq OPT(G) + p$. By Proposition 2.2 the only selfish job on m_i are p-jobs. Also, since the leader schedule the jobs in a sub-optimal assignment then there is at least one selfish job j on m_i. By the pigeonhole principle there is a machine $m_{i'}$ with $L_{i'} < OPT(G)$. Thus, j would benefit from migrating to $m_{i'}$ and s is not stable. ∎

III. Instances with Arbitrary Job Sizes

Algorithm 1 presents a strategy for the leader, given the job sizes and the fraction of load $0 \leq \alpha \leq 1$ that the leader may control. Recall that $P = \sum_j p_j$. Thus, the leader may choose to assign any subset of jobs of total load at most αP. We assume that the leader has unlimited computational power, in particular, it may solve NP-hard problems.

Algorithm 1 - A strategy for assigning the controlled jobs on m machines

1: Sort the jobs such that $p_1 \geq \cdots \geq p_n$
2: Let k be the maximal index for which $\sum_{j \leq k} p_j \leq \alpha P$.
3: Assign the k first jobs in a way that minimizes the maximal load on a machine.

Claim 3.1: Algorithm 1 gives a $(1 + \frac{1 - \frac{1}{m}}{1 + \lfloor \frac{k}{m} \rfloor})$-approximation.

Proof: The analysis is similar to the analysis of the PTAS for the minimum makespan problem [9]. Let s' be the schedule of the k longest jobs and let s denote the NE profile after the selfish jobs join. If $cost(s') = cost(s)$ then s is optimal for G. Otherwise, $cost(s) \geq OPT(G)$.

Let j be the job determining $cost(s)$. Since s is NE, every machine has load of at least $cost(s) - p_j$. Therefore, $P \geq m(cost(s) - p_j) + p_j$. Also, since the jobs are sorted in nonincreasing order of processing times, we have that $p_j \leq p_{k+1}$ and therefore, $P \geq m \cdot cost(s) - (m-1)p_{k+1}$. Furthermore, a lower bound for the optimal solution is a perfectly balanced schedule, thus, $OPT(G) \geq \frac{P}{m}$, which implies that $cost(s) \leq OPT(G) + (1 - \frac{1}{m})p_{k+1}$.

Next we bound p_{k+1} in terms of $OPT(G)$. To obtain such a bound, consider the $k+1$ longest jobs. In an optimal schedule, some machine is assigned at least $\lceil \frac{k+1}{m} \rceil \geq 1 + \lfloor \frac{k}{m} \rfloor$ of these jobs. Since each of these jobs has processing time at least p_{k+1}, we conclude that $OPT(G) \geq (1 + \lfloor \frac{k}{m} \rfloor)p_{k+1}$, which implies that $p_{k+1} \leq \frac{OPT(G)}{1 + \lfloor \frac{k}{m} \rfloor}$ and finally,

$$cost(s) \leq OPT(G)(1 + \frac{1 - \frac{1}{m}}{1 + \lfloor \frac{k}{m} \rfloor})$$

∎

Our next results identify the threshold fraction α, such that controling less than fraction α may not be helpful at all while controlling at least fraction α is guaranteed to be beneficial.

Theorem 3.2: If $\alpha < \frac{1}{m+1}$ then $PoAL(\mathcal{G}, \alpha) = PoA(\mathcal{G})$.

Proof: Recall that $PoA(\mathcal{G}) = 2 - \frac{2}{m+1}$. We show that there exists a game G for which $PoAL(G, \alpha) = 2 - \frac{2}{m+1}$ for every $\alpha < \frac{1}{m+1}$.

Given m, the set of players in G consists of $m^2 - m$ jobs of size 1 and two jobs of size m. Thus, $P = m^2 + m$. $OPT(G) = m + 1$ achieved by assigning the two long jobs on different machines and balance the load with the unit jobs.

Assume the leader controls a fraction $\alpha < \frac{1}{m+1}$ of the load. Thus, it controls load less than m, implying that it controls at most $m - 1$ unit jobs. Therefore, after the leader schedules the controlled jobs, there is an empty machine m_1.

In a possible NE schedule s the two m-jobs are assigned on m_1 and all of the unit jobs are balanced on the remaining machines, each having load m.

We show that s is NE. Clearly, none of the unit jobs on machine $i \neq 1$ may benefit from a migration since $L_i(s) = m$ and $L_1(s) = 2m$. Moreover, since for every $i \neq 1$ $L_i(s) = m$ and there are only two m-jobs scheduled on m_1, they will not benefit from a migration.

Therefore, for any $\alpha < \frac{1}{m+1}$ we have a NE s with $cost(s) = 2m$. Thus, $PoAL(G, \alpha) = \frac{2m}{m+1}$. ∎

In order to show that controlling a fraction $\alpha \geq \frac{1}{m-1}$ guarantees a reduced equilibrium inefficiency, we characterize instances that achieve the worst PoA. We first show that every optimal solution must be perfectly balanced and then that there are at least m relatively short jobs.

Claim 3.3: For every game G with $PoA(G) = \frac{2m}{m+1}$, $OPT(G) = \frac{P}{m}$.

Proof: Let G be a game with $PoA(G) = \frac{2m}{m+1}$ and let $r = \frac{OPT(G)}{m+1}$. Let s be a schedule such that $cost(s) = 2mr$. Therefore, there is a machine M_a with $L_a(s) = 2mr$.

Assume by contradiction that $OPT(G) > \frac{P}{m}$, thus, $P < r(m^2+m)$ then $\sum_{i \neq a} L_i(s) < r(m^2-m)$. Therefore, there is a machine $b \neq a$ with $L_b < rm$. Also, since $L_a(s) = 2mr$ and $OPT(G) = r(m+1)$ it must be that M_a processes at least two jobs and the shortest job on M_a has size at most rm. By migrating to M_b, the shortest job on M_a will reduce its cost below $2rm$, contradicting the stability of s. ∎

Theorem 3.4: If $\alpha \geq \frac{1}{m}$ then $PoAL(\mathcal{G}, \alpha) < 2 - \frac{2}{m+1}$.

Theorem 3.5: If the leader controls m jobs, $PoAL(\mathcal{G}) < PoA(\mathcal{G})$.

Proof: Let G be a load balancing game with $PoA(G) = 2 - \frac{2}{m+1}$ and $m \geq 3$ and let s be the schedule for which $cost(s) = (2 - \frac{2}{m+1})OPT(G)$. Denote i to be a machine with $L_i(s) = cost(s)$. Also, $L_i(s) - OPT(G) = (2 - \frac{2}{m+1})OPT(G) - OPT(G) = (1 - \frac{2}{m+1})OPT(G)$ and for $m \geq 3$ we get $L_i(s) - OPT(G) \geq \frac{OPT(G)}{2}$. Since $L_i(s) > OPT(G)$ there must be a machine i' with $L_{i'}(s) < OPT(G)$. Therefore, i contains two jobs with $p_j > \frac{OPT(G)}{2}$, since otherwise, s is not NE.

Moreover, there are at most m jobs with $p_j > \frac{OPT(G)}{2}$ since otherwise, for every schedule s' there must be a machine i' with $L_{i'}(s') > OPT(G)$ and the optimal value of G can not be achieved. Contradicting the definition of $OPT(G)$.

Let A be the group of jobs with $p_j > \frac{OPT(G)}{2}$. If the leader controls A, since $|A| \leq m$, it may schedule each job on a different machine. Let s' be the resulted schedule s' after the selfish jobs reach NE.

We show that the schedule s' is a NE for a game with all selfish jobs. Assume there is a job j scheduled on machine a that may benefit from a migration. Then since all of the jobs with $p \leq \frac{opt(G)}{2}$ are satisfied, it must be that $p_j > \frac{OPT(G)}{2}$ and there is a machine b with $L_b(s') < \frac{OPT(G)}{2}$. Also, machine b has an additional job j' scheduled on it, since otherwise j is the only job on a and it may not benefit from a migration. It must be that $p_{j'} \leq \frac{OPT(G)}{2}$ since the leader scheduled all of the larger jobs on distinct machines. Thus, j' may benefit by migrating to b. Contradicting the stability of the jobs with sizes smaller than $\frac{OPT(G)}{2}$.

Since s' is a NE then it must be that $cost(s') \leq (2 - \frac{2}{m+1})OPT(G)$. Also, we showed that in every NE with cost $(2 - \frac{2}{m+1})OPT(G)$ it must be that there is a machine with two jobs larger than $\frac{OPT(G)}{2}$. Thus, since s' does not satisfy the condition, $cost(s') < (2 - \frac{2}{m+1})OPT(G)$. ∎

Algorithm 2 presents a strategy for the leader, given a fraction α, which is the load of jobs the leader control.

Algorithm 2 - A strategy for finding an approximation for a schedule with m machines

1: Sort the jobs such that $p_1 \geq \cdots \geq p_n$
2: Assign the k first jobs such that $\sum_{i=1}^{k} p_i \leq \alpha P$ in First-Fit with limit of C^*.

Theorem 3.6: For every $\alpha \leq \frac{m}{m+1}$, Algorithm 2 gives $PoAL(\mathcal{G}, \alpha) \geq 2 - \frac{2}{m+1}$.

Proof: Given m, let G be a game for which the set of players in G consists of m jobs of size m and m jobs of size 1. Thus, $OPT(G) = m+1$ is achieved in a balanced schedule and $PoA(G) = 2 - \frac{2}{m+1}$. Let $\alpha \leq \frac{m}{m+1}$ be the fraction of controlled load. Therefore, the leader controls at most $m-1$ jobs with size m.

If $C^* \geq 2m$, if the leader controls only one job, then the cost after the selfish jobs join may be the worst case. Otherwise, the leader assigns at least two m-jobs on a machine. in both case $cost(s) \geq 2m$ and $PoAL(G, \alpha) \geq 2 - \frac{2}{m+1}$.

If $C^* < 2m$ the leader assigns only one m-job on at most $m-1$ machines. Let M_1 be one of those machines. In a possible NE schedule s a selfish m-job is assigned on M_1, the selfish m-jobs are assigned on different machines and all of the unit jobs are assigned on a single machine. Therefore, each machine having load m except M_1, with $L_1(s) = 2m$. Thus, $PoAL(G, \alpha) = 2 - \frac{2}{m+1}$. ∎

IV. Our Heuristics

In this section we describe the heuristics we have designed and implemented. An instance in our experiments is characterized by a set of jobs, a number of machines and two different heuristics. The first defines the leader's strategy and depends on the fraction of load it controls and the second defines the selfish players behaviour. We describe each of these two classes of heuristics separately.

A. Leader's Strategy

A leader's strategy consists of two steps: (i) Choosing the controlled jobs, (ii) Scheduling the controlled jobs. In this section we describe the heuristics we propose for these steps. Note that the leader's strategy is independent of the selfish jobs' behaviour, however, the leader may benefit and adjust its heuristic if it is known in advance how the selfish jobs will act after it is done assigning the controlled jobs.

1) Choosing the Controlled Jobs: Given a fraction $0 \leq \alpha \leq 1$, the leader needs to choose the jobs it controls. Recall that $P = \sum_j p_j$. Given $0 \leq \alpha \leq 1$, the leader may control jobs of total load at most αP. We implemented three simple heuristics. In the first heuristic, denoted *Pick Smallest*, the leader sorts the jobs in non-decreasing order of size and adds jobs according to this order as long as their total size is at most αP.

In the second heuristic, the leader sorts the jobs in non-increasing order. For choosing the controlled jobs there are two options: it may choose the largest jobs prefix of this sorted set whose total size is at most αP and stop when the next job can not fit, we denote this heuristic *Pick Largest - Stop*, or it may add jobs according to the sorted order and skip jobs whose addition will make the total load more than αP, we denote this strategy *Pick Largest - Skip*.

For example, assume the jobs sizes are $\{1, 2, 3, 4\}$ and let $\alpha = \frac{1}{2}$. Using *Pick Smallest*, the leader controls jobs of sizes $\{1, 2\}$, in *Pick Largest - Stop*, it controls only the job of size 4 and in *Pick Largest - Skip* it controls jobs of sizes $\{4, 1\}$.

2) Leader's Scheduling Strategy: Given the set of controlled jobs, the leader's next mission is to schedule them on the machines. We assumed the leader has limited computational power, meaning, it cannot solve NP-hard problems and in particular, it cannot calculate an optimal schedule for the jobs.

The first algorithm we implemented returns an LPT schedule. Meaning, the leader first sorts the controlled jobs in a non-increasing order. Next, it assigns one job at a time to a least loaded machine. We denote this heuristic *Leader's LPT*.

The second algorithm uses the bin packing algorithm *First Fit*. We denote this heuristic *Leader's FF*. The bin packing problem is an optimization problem, in which items of different sizes must be packed into a finite number of bins or containers, each of a fixed given capacity, in a way that minimizes the number of bins used. First Fit algorithm packs each item into the first bin where it fits, possibly opening a new bin if the item cannot fit into any currently open bin.

We consider the machines as bins and the jobs as items. Let $L = \frac{\sum_{j\in\mathcal{J}} p_j}{m}$. Clearly, L is a lower bound on OPT. In our heuristics, the bins' capacity is determined to be γL for different values of $\gamma \in \{0.9, 1, 1.1\}$. The leader sorts the job on non-increasing order and then uses First Fit with the chosen bins capacity to schedule the controlled jobs on the machines. If some job cannot fit into any machine without an overflow beyond γL (this may happen if α is relatively large), then the jobs is assigned on a lightly loaded machine.

B. Selfish Player's Strategies

The selfish players behaviour also has significant impact on the results. We considered two methods according to which the selfish players reach a NE.

In the first method the selfish players are added to the game sequentially, each added to a least loaded machine at that time.

The jobs are considered in LPT order. We denote this strategy *Player's LPT*. We claim that in this case, after adding the selfish players, the schedule is stable against deviation of the selfish players. The proof of the following claim follows from the analysis of LPT for classical load balancing games [21].

Claim 4.1: LPT algorithm for non-empty machines produces a NE for the selfish jobs.

In the second method the jobs are added greedily in arbitrary order and then we use *Best Response Dynamics (BRD)*. The best response dynamics is one of the most elementary methods in game theory. Using Round Robin according to the jobs assignment order. Each player at its turn tries to improve its state. If there is a machine for which the player may benefit from a migration, it will migrate to a most beneficial option. Otherwise, it stays on the same machine. The algorithm stops when non of the players has a beneficial migration. It is known that BRD algorithm converges to a NE when starting the algorithm with empty machines. The discussed case of adding players to non-empty machines is a sub-problem of it and achieves NE similarly.

V. Experimental Results

Our instances consist of 20 machines and 100 jobs. It is common to use exponential times for job processing times as it is often a good approximation of service times [19]. In order to work with integers, we used geometrical distribution and chose parameter 0.13 which provides instances with mostly small jobs, with an amount of larger jobs that allows many combinations of equilibrium with different makespan values. Diversity of job sizes and relatively low loads, enables a good comparison between different heuristics.

The random jobs generator may create jobs whose size is larger than the optimal makespan, calculated by $\frac{\sum_{j\in\mathcal{J}} p_j}{m}$. We address these jobs as outliners. The presence of such jobs hurts the comparison of the different heuristics, since they may cause that an instance will have the same makespan for all strategies. By scheduling one such long job on a machine, non of the selfish jobs may benefit by choosing this machine. In order to emphasis the differences between the different heuristics, we remove the outliners using the following method. We note that on average $1-2$ jobs were removed from each instance.

Algorithm 3 - Removing outliners

1: Calculate $avgLoad(G) = \frac{\sum_{j\in\mathcal{J}} p_j}{m}$
2: Remove every job $j \in \mathcal{J}$ with $p_j > avgLoad(G)$.
3: Repeat steps $1, 2$ while some job was removed.

In the following sections we present our experimental results. These results were obtained by running our heuristics on the same 100 instances, each consisting of 100 players and 20 machines. For each instance we calculated the lower bound, $L = \sum_j p_j/M$, on the optimal makespan. For each experiment we present in the diagrams the average makespan scaled by L. For example, if the average makespan in all the runs performed in some experiment is $1.2L$ then the corresponding bar in the figure has height 1.2. Since $L \leq OPT$, this ratio is an upper bound on the price of anarchy.

A. Choosing the Controlled Jobs

In this section, we present our results for the comparison of the first part in the leader's strategy. Specifically, we compare how the performance is affected by the fraction of load controlled by the leader and the different heuristics for choosing the leader's controlled jobs. We consider all the possible combinations of leader and selfish players strategies. Recall that *Leader's FF* algorithm schedules each job on the first machine where it fits with limit of $\gamma\frac{\sum_{j\in\mathcal{J}} p_j}{m}$ on the totals' machine load for $\gamma \in \{0.9, 1, 1.1\}$, possibly starting to schedule on a new machine if the job cannot fit into any currently open machine. In the experiment described below, we fixed $\gamma = 1$ as a representative case for the strategy.

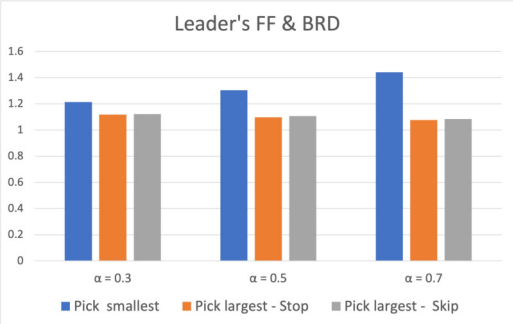

Fig. 3. Comparing jobs choosing strategies for Leader's FF with BRD

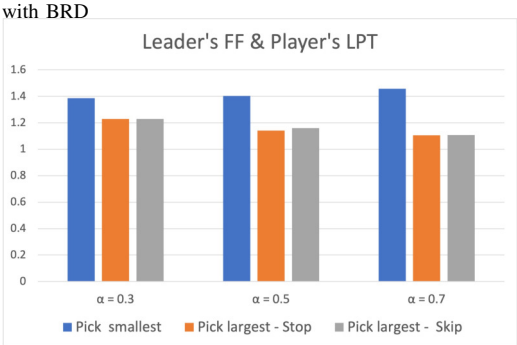

Fig. 4. Comparing jobs choosing strategies for Leader's FF with Player's LPT

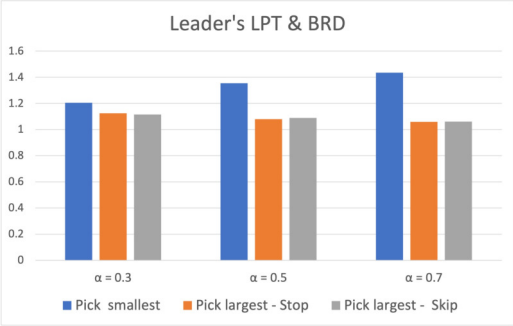

Fig. 5. Comparing jobs choosing strategies for Leader's LPT with BRD

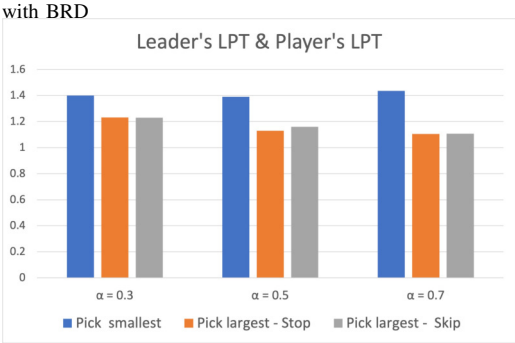

Fig. 6. Comparing jobs choosing strategies for Leader's LPT with Player's LPT

Figures 3,4,5 and 6 present the differences between the possible leader's strategies for choosing controlled jobs for every possible combination of considered strategies for the leader and selfish players. The figures present the PoA (y axis) as a function of α for every strategy.

First, we can conclude that *Pick Smallest* provides the worst results for all α values and heuristics. As shown in the figures, the PoA increases with α, meaning that the leader's control hurts the social cost.

In contrast, we can also conclude that for the other two strategies *Pick Largest - Skip* and *Pick Largest - Stop*, as α increases, the final schedule is better and the leader reduces the makespan in average by factor $1.16, 1.11, 1.08$ if it controls $0.3, 0.5, 0.7$ fraction of the load respectively. Other than that, the difference between both strategies is minor, with a slight advantage to the *Pick Largest - Stop* strategy in most of the cases.

B. Leader's Scheduling Strategy

In this section we discuss the optimal strategy we recommend the leader to apply on the controlled jobs. The best strategy may differ between the amount of load the leader controls, the selfish players behaviour or other parameters we examine.

The experiments that consider the selection of the controlled jobs, reported in section 5.1, reveal that *Pick Largest - Stop* has the best performance, thus, we use it in the following experiments.

1) Leader's First Fit: Our next experiments analyze the influence of the parameter γ in the *Leader's FF* strategy. Recall that the *Leader's FF* strategy schedules the controlled jobs using *First Fit* algorithm with machine capacity of $\gamma \sum_j \frac{p_j}{m}$ for $\gamma > 0$. We applied this strategy with $\gamma \in \{0.9, 1, 1.1\}$. These values were chosen since we aim to have approximate PoA closer to 1; if we choose a low γ value, the selfish players may determine the makespan since larger jobs may join non-empty machines.

If we choose higher γ value, then already the jobs assigned by the leader may cause a high makespan. On the other hand, this leaves more empty machines for the selfish jobs and potentially prevents them from reaching load higher than $\gamma \sum_j \frac{p_j}{m}$ on the reamaining machines.

Figures 8 and 7 present the approximated PoA with regards to the discussed options for γ parameter and the controlled fraction α achieved with $\alpha \in \{0.3, 0.5, 0.7\}$ and $\gamma \in \{0.9, 1, 1.1\}$. In the experiments described in Figure 7 the selfish jobs join and perform BRD. In the experiments described in Figure 8 they are added using LPT rule.

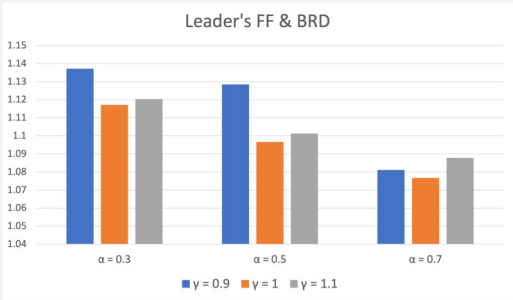

Fig. 7. Comparing γ parameter for *Leader's FF* strategy with *BRD*

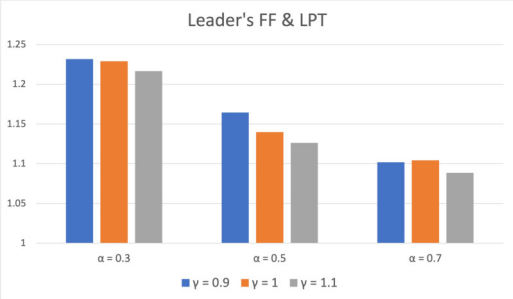

Fig. 8. Comparing γ parameter for *Leader's FF* strategy with *Selfish LPT*

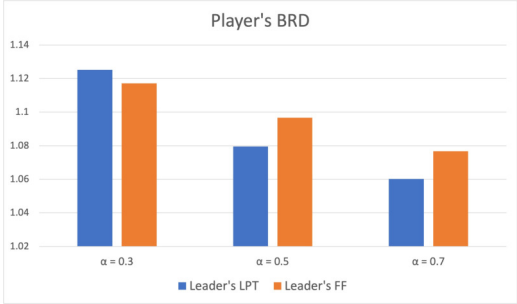

Fig. 9. Comparing leader's possible strategies for Player's BRD

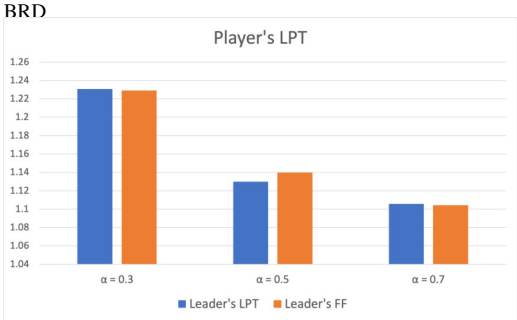

Fig. 10. Comparing leader's possible strategies for Player's LPT

As presented in Figure 7, when the selfish players use *Player's BRD* strategy, for every fraction α, the best performance is achieved when $\gamma = 1$. For $\gamma = 0.9$ we have significantly worst result for $\alpha \in \{0.3, 0.5\}$, but better than $\gamma = 1.1$ for $\alpha = 0.7$. This difference is explainable since the leader have the most significant control for this α and both approximated PoA results values are below 1.1 and for $\gamma = 1.1$ the leader schedules as much as it can near the load of $1.1 \sum_j \frac{p_j}{m}$ which ensured result near 1.1. In contrast to $\gamma = 0.9$ which allowed lower results.

On the other hand, for *Player's LPT* in Figure 8 we get consistent results for all α values. We get the worst results for the value $\gamma = 0.9$. Next, we have $\gamma \in \{1, 1.1\}$ with close results, with a minor advantage to $\gamma = 1.1$.

2) Leader's Strategy: We conclude with experiments that compare the leader's strategy choice with regards to the selfish players behaviour. Based on the results of thee previous experiments, in all the experiments we report in this section, the controlled jobs are chosen using *Pick Largest - Stop* and for *Leader's FF* we use parameter $\gamma = 1$. In the experiments presented bellow, we compared which leader's strategy gives better results with respect to the applied player's strategy for every α choice.

The results shown in both Figures 9 and 10, imply that for $\alpha = 0.3$ we get better makespan when using *Leader's FF* strategy. On the other hand, for larger α we get that *Leader's LPT* perform better. Moreover, we can conclude that when the selfish players use BRD, the difference between the makespans resulting from the two leader's strategy are more significant comparing to games in which the selfish players are added by LPT.

Another interesting result we can infer from these experiments, is the difference between the results for selfish players behaviour. For all α values and leader's strategy choice, we get much lower makespans when the selfish players use BRD.

VI. Conclusions

In this work we examined the potential influence of a leader who control a subset of the jobs in load balancing games. The leader assigns the jobs it controls and the other jobs then select their assignments in a selfish way. We present theoretical as well as experimental results answering several questions regarding the advantages and disadvantages of controlling a fraction of the load. We have designed and implemented several heuristics for possible strategies the leader may apply when choosing the jobs to control and when scheduling them on the machines.

In the theoretical results, we proved that if the leader is obligated to control a maximal amount of the given fraction, the resulting schedule may be worse than a NE schedule in a leader-free environment. On the other hand, for a setting in which the leader may choose to exploit only part of its power, we presented tight bounds on the fraction α of the controlled load that may improve the game cost. Additionally,

we presented several relations between the common $PoA(\mathcal{G})$ measure and the $PoAL(\mathcal{G}, \alpha)$ measure which we defined.

Our experiments show that controlling the largest jobs in a game is the best strategy for the leader, while controlling the smallest jobs may cause significant damage. In fact controlling no job may be better than controlling only small ones. One of the main conclusions that resulted from the experiments is that controlling a larger fraction of the load improves the game cost when controlling the larger jobs.

We have observed that when using the heuristic *Leader's FF*, the performance is significantly influenced by the choice of the capacity to which the machines are loaded. Let $L = \sum_j \frac{p_j}{m}$, then for a parameter γ, the leader adds jobs to machine in a First-Fit matter, where each job is added to the first machine whose addition will result in total load at most γL. In general, the best result achieved for this strategy is when $\gamma = 1$. Although, if the selfish players scheduling strategy is known to be LPT, then filling the machines up to capacity $1.1L$ results is slightly better outcomes. Also, for large α values the results for all γ values are very close, but on the other hand for low α we can definitely conclude that we should not use γ less than 1. Future work on this algorithm may find more characteristics for choosing the γ value.

Another conclusion we can infer, is that knowing the selfish player's behaviour in advance may be helpful for the leader. Similarly, the strategy choice depends also on α - the fraction of controlled jobs. In general, for both selfish players methods it holds that the *Leader's LPT* strategy gives the best results, but if it is known that the selfish players acts in BRD and the controlled fraction is low we may consider using *Leader's FF*.

For future work it may be interesting to improve the *Leader's FF* strategy and characterize the jobs that the leader should choose specifically for using this heuristic. *Leader's FF* is a heuristic in which the leader assigns the controlled jobs on a subset of the machines and leave some of the machines empty for the selfish jobs. We believe that additional algorithms using this approach should be considered and analyze and may perform even better than *Leader's FF*. Also, it may be interesting to get inspiration from additional known approximation algorithms for the minimum makespan problem as we did with *Leader's LPT* which is based on the algorithm presented in [7].

In the theoretical side, we believe that there are many interesting open problems in this setting. For example, analyzing weighted Stackelberg games of congestion games with non-singelton strategy space, non symmetric singelton games, or games with non-uniform resources, that is, machines with different speeds. Also, in our settings the leader has unlimited computational power, meaning, it is able to solve NP-hard problems. It will be interesting to have results for a leader with a limited computational power. Another interesting direction is to analyze the consequences of controlling a specific given subset of jobs $A \subseteq \mathcal{J}$ instead of a fraction α of the load.

REFERENCES

[1] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. *SIAM Journal on Computing*, 38(4):1602–1623, 2008.

[2] V. Bonifaci, T. Harks, and G. Schäfer. Stackelberg Routing in Arbitrary Networks. Mathematics of Operations Research, 35(2), 330–346, 2010.

[3] V. Bilò and C. Vinci. On stackelberg strategies in affine congestion games. *Theory of Computing Systems*, 63(6):1228–1249, 2019.

[4] Y. Cho and S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9(1):91–103, 1980.

[5] A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. *ACM Trans. Algorithms*, 3(1):4:1–4:17, 2007.

[6] L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. In *Proc. of the 7th European Symposium on Algorithms(ESA)*, 1999.

[7] G. Finn and E. Horowitz. A linear time approximation algorithm for multiprocessor scheduling. *BIT Numerical Mathematics*, 19(3):312–320, 1979.

[8] D. Fotakis. Stackelberg strategies for atomic congestion games. *Theory of Computing Systems*, 47(1):218–249, 2010.

[9] M.R. Garey and R.L. Graham. Bounds for Multiprocessor Scheduling with Resource Constraints. *SIAM J. J. Comput.*, 4(2): 187–200, 1975.

[10] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.

[11] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17:263–269, 1969.

[12] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: Practical and theoretical results. *Journal of the ACM*, 34(1):144–162, 1987.

[13] S. G Karakostas, G. Kolliopoulos. Stackelberg strategies for selfish routing in general multicommodity networks. *Algorithmica*, 2009.

[14] H. Kellerer and U. Pferschy. Cardinality constrained bin-packing problems. *Annals of Operations Research*, 92:335–349, 1999.

[15] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. *STACS*, pages 404–413, 1999

[16] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009.

[17] VS A. Kumar and M. V Marathe. Improved results for stackelberg scheduling strategies. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2002.

[18] T. Roughgarden. Stackelberg scheduling strategies. *SIAM Journal on Computing*, 33(2):332–350, 2004.

[19] P. M. Swamidass (Eds.), Exponential service times. In *Encyclopedia of Production and Manufacturing Management*, 2000.

[20] A. S. Schulz and N. E Stier Moses. On the performance of user equilibria in traffic networks. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 86–87, 2003.

[21] B. Vöcking. *Algorithmic Game Theory*, chapter 20: Selfish Load Balancing. Cambridge University Press, 2007.