On the Price of Anarchy of Restricted Job Scheduling Games

Diodato Ferraioli* and Carmine Ventre**

Abstract. The Price of Anarchy [Koutspoupias and Papadimitriou, 1999] is the, by now celebrated, ratio used to measure how system performance is affected by the selfish behaviour of agents using it. In this work we show bounds on the Price of Anarchy for problems of restricted scheduling of selfish jobs on related and unrelated machines.

1 Introduction

In last ten years Game Theory emerged as a fundamental tool to study the complex interactions among autonomous and selfish agents in a complex system, such as Internet, P2P networks, social networks. Nash equilibrium is one of the basic concepts of Game Theory and represents a "stable" configuration where each agent is not interested in unilaterally change her behaviour, given the actions of other agents. Thus, a Nash Equilibrium (NE) configuration has the interesting property to be stable even if agents are autonomous and selfish and cannot be forced to follow a given protocol. Game theorists have deeply studied the problems of existence and structure of Nash Equilibria, e.g. the celebrated Nash Theorem guarantees that each finite game has a mixed NE. A well-known question about NE is that a game can have more NE and it is not possible to foresee which equilibrium the agents will reach. In [10] Koutsoupias and Papadimitriou dealt with the problem of the efficiency of NE using the classical computer science worst case approach. They defined the *Price of Anarchy* (PoA) as the ratio between the worst NE and the centrally computed solution, optimal for some global objective function referred to as *social function*. This quantity gives a measure of the loss of performance due to the lack of coordination among the agents and their selfishness. Roughly speaking, a low PoA means that agents behaviour have a small impact on the efficiency of the system; an high PoA, instead, gives evidence that agents can really hurt the system and some sort of "coordination" is useful. In last years several researchers analyzed the PoA for several class of games, such as selfish routing [12], job scheduling [10, 6], network formation games [8, 1] and many others (see [11] for a complete picture).

In [10], authors focused on a simple, but very interesting, model, called KP-model. In the KP-model one is given a network of m parallel links, with the same

* Dipartimento di Informatica ed Applicazioni, Universit di Salerno, Italy. Email: ferraioli@dia.unisa.it.

** Computer Science Department, University of Liverpool, UK. Email: Carmine.Ventre@liverpool.ac.uk.

capacity, and n selfish agents that have to transfer w_i amount of traffic while trying to minimize their own latency. To do so, agents can strategically choose the link to use to deliver their traffic. The social function of interest is instead the minimization of the maximum latency suffered by the players. This problem can also be viewed as the minimization of the makespan in a job scheduling setting with m identical machines and n independent and selfish jobs of length w_i. There are several more problems that are well-represented by this model: Internet Service Provider selection in a business network, server selection in a web server farm [5], peer selection in a P2P network [14] and so on.

In this work, we extend the KP-model, considering the case of restricted scheduling. In our setting each job has a set of allowed machines where it can be assigned. We consider both the related (every machine has a different speed to execute jobs) and unrelated (every machine has job-dependent speed) case. An immediate example of our restricted model is the access point selection in a wireless network. Indeed, a user can connect only to access points that are in its transmission range.

We propose bounds on the PoA in such a model with respect to several different social functions.

The model. We have n jobs that have to be executed by m machines, and every job i has a subset A_i of allowed machines. We denote by $w_{ij} \in [w_{min}, w_{max}]$, $w_min > 0$, the load that job i carries on machine j. In the *related* machines model, the weights of the jobs are independent from the machines, but each machine j has a speed v_j and so in this case $w_{ij} = \frac{w_i}{v_j}$, where w_i denotes the machine-indepedent weight of job i. We define $s = \frac{w_{max}}{w_{min}}$. In the related machines model it will be useful to distinguish the following ratios $t = \frac{\max_i w_i}{\min_i w_i}$ and $r = \frac{\max_j v_j}{\min_j v_j}$.

A schedule S assigns each job to only one of its allowed machines. Given S, we define $j_i(S)$ as the machine where the job i is assigned in S, $n_j(S)$ as the number of jobs that S assigns to the machine j, $c_j(S) = \sum_{i:j_i=j} w_{ij}$ as the total load on the machine j in S. We assume that the latency suffered by the job i in S is $c_{j_i(S)}(S)$ as in [10]. This assumption that can be justified by machines scheduled in a round-robin fashion for small enough quanta.

A schedule S is a Nash solution if

$$\forall i, \forall j \in A_i, c_{j_i(S)}(S) \leq c_j(S) + w_{ij}.^1 \tag{1}$$

(Below when we refer to a Nash solution we use the simplified notation j_i, n_j and c_j.)

A schedule S is optimal if it minimizes a given social function. We are interested here in three different kind of social functions: (i) the *Server Latency*

[1] Observe that we focus on so-called *pure* NE. Since pure equilibria are more natural than mixed ones, it is preferred to consider PoA with respect to pure NE for games possessing them. Our scheduling games always possess pure NE [7]. So, in the sequel when we say NE we actually mean pure NE.

that minimizes the total work of the machines, i.e., $\min_S \sum_j c_j(S)$; (ii) the *Total Latency* that minimizes the sum of the latencies suffered by the jobs, i.e., $\min_S \sum_i c_{j_i(S)}(S) = \min_S \sum_j n_j(S)c_j(S)$ and (iii) the *Weighted Total Latency* that minimizes the weighted sum of the latencies suffered by the jobs, i.e., $\min_S \sum_i w_{ij_i(S)} c_{j_i(S)}(S)$. (Below to ease our notation when we refer to an optimal solution we use the simplified notation j_i^*, n_j^* and c_j^*.)

Related Work. A huge literature emerged in the last years with respect to the subject of efficiency of games equilibria. Here we focus only on results regarding job scheduling (we refer to [11] for a more wide overview).

Even-dar *et al.* in [7] show that a NE always exists in job scheduling with unrelated machines.

Several results are known about the PoA with respect to different social functions. Awerbuch *et al.* in [3] study the efficiency of NE with respect to Maximum Latency and prove that $PoA = \Theta(\frac{\log m}{\log \log m})$ for related machines and $PoA = \Theta(s + \frac{\log m}{\log(1+\frac{\log m}{s})})$ for unrelated machines.

Another widely studied social function is the Weighted Total Latency. With respect to this, Awerbuch *et al.* [2] prove an upper bound of $\frac{3+\sqrt{5}}{2} \approx 2.618$ on PoA for the more general case of routing on a network with affine latency functions and weighted jobs, that holds also in the case of job scheduling with related and identical machines. Caragiannis *et al.* [4] prove that this bound is tight in related machines case while there exists a lower bound of 2.5 for the case of identical machines: it is an open problem to close the gap between upper and lower bound in this last case.

With respect to Total Latency, Suri *et al.* [13] show that $PoA \le 2.5$ when machines are related and all jobs have same weight and Caragiannis *et al.* [4] show a tight lower bound for the same setting; Hoefer e Souza in [9] show that if machines are identical and 0-weight job are allowed, then $PoA \le \frac{n\sqrt{m}}{w} + \frac{2nm^2}{w^2}$, where $w = \sum_i w_i$.

Our Results. In our work we present first results in a restricted setting where neither machines nor jobs are all identical. We can summarize our results with respect to the relative social function: for the Server Latency social function, we show tight bounds on the PoA: $PoA = \Theta(s)$ in the unrelated machines model and $PoA = \Theta(\min(r, n))$ in the related machines model; for the Weighted Total Latency social function we extend the bound of [2] to the unrelated machines setting, showing that $PoA = \Theta(s^2)$; for the Total Latency social function we show an upper bound of ms on the PoA and a lower bound of s in the unrelated machines setting and of $\min\left(\frac{m+\sqrt{t}}{m+3}, \frac{n}{m+3}\right)$ in the related machines setting. Proofs and technicalities can be found in appendix.

Note that no results were known for the Server Latency social function. We believe that such a social function has interesting practical applications. Server latency in fact measures how bad is the resource usage for the service provider due to selfishness. For example, in a server farm setting this social function measures how servers work and gives useful suggestions to farm maintainer.

2 Main Theorems

Theorem 1. *With respect to Server Latency social function, job scheduling game with unrelated machines has* $PoA = \Theta(s)$.

Theorem 2. *With respect to Server Latency social function, job scheduling game with related machines has* $PoA = \Theta(\min(r, n))$.

Theorem 3. *With respect to Weighted Total Latency objective function, job scheduling game with unrelated machines has* $PoA = \Theta(s^2)$.

Theorem 4. *With respect to Total Latency social function, job scheduling game has* $PoA = O(ms)$.

Theorem 5. *With respect to Total Latency social function, job scheduling game with related machines has* $PoA = \Omega\left(\min\left(\frac{m+\sqrt{t}}{m+3}, \frac{n}{m+3}\right)\right)$.

Open Problems. It is very interesting to find tight bounds for PoA with respect to Total Latency social function: we think that real bounds are closer to our lower bounds than to our upper bounds.

References

1. Susanne Albers, Stefan Eilts, Eyal Even-Dar, Yishay Mansour, and Liam Roditty. On nash equilibria for a network creation game. In *SODA*, pages 89–98, 2006.
2. Baruch Awerbuch, Yossi Azar, and Amir Epstein. The price of routing unsplittable flow. In *STOC*, pages 57–66, 2005.
3. Baruch Awerbuch, Yossi Azar, Yossi Richter, and Dekel Tsur. Tradeoffs in worst-case equilibria. *Theor. Comput. Sci. 361(2-3)*, pages 200–209, 2006.
4. Ioannis Caragiannis, Michele Flammini, Christos Kaklamanis, Panagiotis Kanellopoulos, and Luca Moscardelli. Tight bounds for selfish and greedy load balancing. In *ICALP (1)*, pages 311–322, 2006.
5. Artur Czumaj, Piotr Krysta, and Berthold Vocking. Selfish traffic allocation for server farms. In *STOC*, pages 287–296, 2002.
6. Artur Czumaj and Berthold Vocking. Tight bounds for worst-case equilibria. *ACM Transactions on Algorithms 3(1)*, 2007.
7. Eyal Even-dar, Alex Kesselman, and Yishay Mansour. Convergence time to nash equilibria. *ACM Transactions on Algorithms 3(3)*, 2007.
8. Alex Fabrikant, Ankur Luthra, Elitza N. Maneva, Christos H. Papadimitriou, and Scott Shenker. On a network creation game. In *PODC*, pages 347–351, 2003.
9. Martin Hoefer and Alexander Souza. The influence of link restrictions on (random) selfish routing. In *SAGT*, pages 22–32, 2008.
10. Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *STACS*, pages 404–413, 1999.
11. Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory.* Cambridge University Press, 2007.
12. Tim Roughgarden and Eva Tardos. How bad is selfish routing? *J. ACM 49(2)*, pages 236–259, 2002.
13. Subhash Suri, Csaba D. Tóth, and Yunhong Zhou. Selfish load balancing and atomic congestion games. *Algorithmica 47(1)*, pages 79–96, 2007.
14. Subhash Suri, Csaba D. Tth, and Yunhong Zhou. Uncoordinated load balancing and congestion games in p2p systems. In *IPTPS*, pages 123–130, 2004.

A Missing Proofs

Throughout the appendix for a given schedule S, we will let $C(S)$ denote the cost of S for the social function of interest. E.g., for Server Latency function, $C(S) = \sum_{j=1}^{m} c_j(S)$.

A.1 Server Latency

First of all we focus on the unrelated machines setting, showing our tight bound for the PoA.

Theorem 1. *With respect to Server Latency social function, job scheduling game with unrelated machines has $PoA = \Theta(s)$.*

Proof. It is trivial to upper bound the Price of Anarchy: in fact, since $w_{ij_i} \leq sw_{ij_i^*}$, it results that $\sum_j c_j \leq s \sum_j c_j^*$.

There is also a simple example to show this result is tight: we have n jobs and $m = n$ machines and every job $i \in \{0, \dots, n-1\}$ can be assigned to machine i with weight 1 and to machine $(i+1) \bmod n$ with weight s. The optimal schedule assigns every job i to machine i and its cost is n. It is to verify that the solution that assigns every job i to machine $(i+1) \bmod n$ is a Nash equilibrium with cost sn. □

We now focus on the related machines model: Lemmata 1 and 2 show Theorem 2, i.e., show that $PoA = \Theta(\min(r, n))$.

Lemma 1. *With respect to Server Latency social function, job scheduling game with related machines has $PoA = O(\min(r, n))$.*

Proof. For every job i we define $j_{i-MAX} = \arg\max_{j \in A_i} v_j$ and for every machine j we define $W_j = \sum_{i:j_i=j} w_i$.

Claim. Assignment S such that $\forall\ i\ j_i(S) = j_{i-MAX}$ is optimal.

Proof. Suppose by contradiction that the claim is not true. This means that we can move some jobs to improve the social cost. But every change moves a job i from machine j_{i-MAX} to a machine $j_i^* \neq j_{i-MAX}$, where $v_{j_i^*} \leq v_{j_{i-MAX}} \Rightarrow \frac{w_i}{v_{j_i^*}} \geq \frac{w_i}{v_{j_{i-MAX}}}$. So, no move improves the cost of S. □

So now we refer OPT as the schedule such that $\forall\ i\ j_i^* = j_{i-MAX}$. From Claim A.1 we derive the following corollaries:

Corollary 1. $C(OPT) = \sum_j c_j^* = \sum_i \frac{w_i}{v_{j_{i-MAX}}}$.

Corollary 2. $\forall\ j$ *such that* $c_j^* \neq 0$, $c_j \leq c_j^*$.

From Nash condition (1) we obtain

$$\frac{w_i}{v_{j_i}} \leq \frac{W_{j_i}}{v_{j_i}} = c_{j_i} \leq c_{j_{i-MAX}} + \frac{w_i}{v_{j_{i-MAX}}} = \frac{W_{j_{i-MAX}} + w_i}{v_{j_{i-MAX}}} \tag{2}$$

But

$$v_{j_i} \geq \min_j v_j = \frac{\min_j v_j}{\max_j v_j} \max_j v_j \geq \frac{1}{r} v_{j_i^*} \tag{3}$$

Merging (2) and (3), we have

$$v_{j_i} \geq \max\left(\frac{v_{j_{i-MAX}} w_i}{W_{j_{i-MAX}} + w_i}, \frac{1}{r} v_{j_i^*}\right) \tag{4}$$

Let d be the max degree of a machine, that is the max number of jobs that can be assigned to the same machine.

If $\max\left(\frac{v_{j_{i-MAX}} w_i}{W_{j_{i-MAX}}+w_i}, \frac{1}{r} v_{j_i^*}\right) = \frac{v_{j_{i-MAX}} w_i}{W_{j_{i-MAX}}+w_i}$, using bound (4) and Corollaries 1 and 2 we have:

$$\begin{aligned} C(NASH) &= \sum_i \frac{w_i}{v_{j_i}} \leq \sum_i \frac{W_{j_{i-MAX}} + w_i}{v_{j_{i-MAX}}} = \sum_i c_{j_{i-MAX}} + C(OPT) = \\ &= C(OPT) + \sum_{j:c_j^* \neq 0} (n_j^* c_j) \leq C(OPT) + d \sum_{j:c_j^* \neq 0} c_j^* = (d+1)C(OPT). \end{aligned} \tag{5}$$

If, instead, $\max\left(\frac{v_{j_{i-MAX}} w_i}{W_{j_{i-MAX}}+w_i}, \frac{1}{r} v_{j_i^*}\right) = \frac{1}{r} v_{j_i^*}$, then, using bound (4) we have:

$$C(NASH) = \sum_j c_j = \sum_i \frac{w_i}{v_{j_i}} \leq r \sum_i \frac{w_i}{v_{j_i^*}} = rC(OPT) \tag{6}$$

So we can deduce that $PoA = O(\min(r, d))$.
Observe that $d = O(n)$, and thus $PoA = O(\min(r, n))$. □

Lemma 2. *With respect to Server Latency social function, there exists a job scheduling game with related machines where* $\forall\, \varepsilon > 0$ $PoA = \Omega(\min(r-\varepsilon, n-\varepsilon))$.

Proof. Let $n = m$. A machine has speed r, others have speed 1. A job has weight r, others have weight 1. Every job can be assigned to the fast machine and only one different slow machine.

The optimal schedule assigns every job to the fast machine, so $C(OPT) = \frac{r+n-1}{r}$.

The Nash solution assigns the heavy job to the fast machine and remaining jobs to different machines, so that every machine has load 1, that is $C(NASH) = n$.

If $\forall\, \varepsilon > 0$ $\frac{r}{r+n-1} \geq 1 - \frac{\varepsilon}{n}$, that is $r \geq \frac{n^2}{\varepsilon}$, and so $\min(n, r) = n$, then

$$PoA \geq n - \varepsilon.$$

If, instead, $\forall\, \varepsilon > 0$ $\frac{n}{r+n-1} \geq 1 - \frac{\varepsilon}{r}$, that is $n \geq \frac{r^2}{\varepsilon}$, and so $\min(n, r) = r$, then

$$PoA \geq r - \varepsilon.$$

□

A.2 Weighted Total Latency

We extend the bound of [2] to the unrelated machines setting. Our result is showed in the Theorem 3.

Theorem 3. *With respect to Weighted Total Latency objective function, job scheduling game with unrelated machines has* $PoA = \Theta(s^2)$.

Proof of the upper bound adapts ideas depicted in [2] to our setting.

Proof. Summing both sides of Nash condition (1) over all job i and multiplying every term by w_{ij_i} we obtain:

$$\begin{aligned} C(NASH) &= \sum_i w_{ij_i} c_{j_i} \leq \sum_i w_{ij_i}(c_{j_i^*} + w_{ij_i^*}) \leq \sum_i w_{ij_i}(c_{j_i^*} + c^*_{j_i^*}) \leq \\ &\leq \sum_i w_{ij_i} c_{j_i} \leq s \cdot \sum_i w_{ij_i^*}(c_{j_i^*} + c^*_{j_i^*}) = s \cdot \sum_j (c_j^*)^2 + s \cdot \sum_j c_j^* c_j = \\ &= s \cdot C(OPT) + s \cdot \sum_j c_j^* c_j \end{aligned} \tag{7}$$

Using *Cauchy-Schwartz inequality*:

$$\sum_j c_j c_j^* \leq \sqrt{\sum_j (c_j)^2} \cdot \sqrt{\sum_j (c_j^*)^2} = \sqrt{C(NASH)} \cdot \sqrt{C(OPT)}. \tag{8}$$

By (7) and (8), dividing all by $C(OPT)$ we obtain that $PoA = \frac{C(NASH)}{C(OPT)} \leq s + s$, where solving equation gives bound:

$$PoA \leq \frac{1}{2}(s^2 + 2s + \sqrt{s^4 + 4s^3}) = O(s^2).$$

□

Moreover, it is easy to see that there exists a job scheduling game with unrelated machines such that $PoA = \Omega(s^2)$. Indeed, in the same simple example used for the lower bound of Theorem 1, the optimal schedule costs n, while the Nash equilibrium solution costs $s^2 n$.

A.3 Total Latency

First of all, we focus on upper bound for the PoA: it is showed for the unrelated machines setting, but it obviously holds for the related machines as well. Our result is pointed out in Theorem 4.

Theorem 4. *With respect to Total Latency social function, job scheduling game has* $PoA = O(ms)$.

In the proof we establish, using the common "*Nash Condition*" argument, that $C(NASH) \leq C(OPT) + \sum_j n_j^* c_j$ and then we try to bound $\sum_j n_j^* c_j$.

Proof. By Nash condition

$$\forall\ i\ c_{j_i} \leq c_{j_i^*} + w_{ij_i^*}$$

Summing both sides over all jobs i and knowing that $w_{ij_i^*} \leq c^*_{j_i^*}$, we have

$$\sum_i c_{j_i} \leq \sum_i (c_{j_i^*} + c^*_{j_i^*})$$

Moving the sum over all job i to a sum over all machines j, we obtain

$$\sum_j n_j c_j \leq \sum_j n_j^* c_j + \sum_j n_j^* c_j^* \Rightarrow C(NASH) \leq C(OPT) + \sum_j n_j^* c_j \tag{9}$$

Now we can upper-bound $\sum_j n_j^* c_j$ as follows:

$$\sum_j n_j^* c_j \leq \sum_j n_j^* n_j w_{max} \leq w_{max} (\sum_j n_j^*)(\sum_j n_j) = n^2 w_{max} \tag{10}$$

At same time, we can easily lower-bound $C(OPT)$ as follows:

$$\sum_j n_j^* c_j^* \geq \sum_j (n_j^*)^2 w_{min} \geq w_{min} \frac{(\sum_j n_j^*)^2}{m} = \frac{n^2}{m} w_{min} \tag{11}$$

So, from (9), (10) and (11) we obtain

$$PoA = \frac{C(NASH)}{C(OPT)} \leq \frac{C(OPT)}{C(OPT)} + \frac{\sum_j n_j^* c_j}{C(OPT)} \leq 1 + m \frac{n^2 w_{max}}{n^2 w_{min}} = O(ms).$$

□

The above result is not tight. In fact, in unrelated machines setting we can only prove a lower bound of s; just consider the same example used for the lower bound of Theorem 1. For the related machines setting we show $PoA = \Omega\left(\min\left(\frac{m+\sqrt{t}}{m+3}, \frac{n}{m+3}\right)\right)$ in Appendix B.

B Proof of Theorem 5

We define the following instance I in which we let $w = \left\lceil \frac{(k-1)(m+3)+3}{2r+m-2} \right\rceil$, where k will be valued below. In the instance I, machines 1 and m have speed r (we will refer sometimes to these as fast machines), while the others have speed 1 (referred to as slow machines). The jobs are partitioned in four sets: A, B, C and D. Set A has size $2\lceil r \rceil w$: each job in the set has weight 1 and can be assigned only to machines 1 and m. We let a denote $|A|$. Set B has size $(m-2)w$. Each job in B weights 1 and can be assigned to all machines. We let b denote $|B|$. Set C has size 2. Each such job has weight $t \geq (a+b)^2$ and can be assigned only to machines 1 and m. Set D has size $m-2$. Each job in D has weight $\frac{t}{r}$. There is a one-to-one mapping between machines $2, \ldots, m$ and jobs in D so that each job in D can only be assigned to one different slow machine.

Consider the following solution: all jobs in A and B are assigned to machine 1; jobs in C are assigned to machine m; each job in D is assigned to its only allowed machine. Consistently with the next lemma we will refer to this solution as OPT.

Lemma 3. *Solution above is optimal for the instance I.*

Proof. To show that OPT is indeed optimal firstly we focus on the fast machines (machines 1 and m) and show what is an optimal schedule for them and then we analyze slow machines (machines $2, \ldots, m-1$).

Since only jobs in B can be assigned to both fast and slow machines, we artificially split the set of jobs B in two subsets B_f and B_s. B_f is the set of jobs in B to be assigned to the fast machines, while B_s contains those jobs in B to be assigned to slow machines. (Note that in OPT, $B_f = B$ and $B_s = \emptyset$.) Extending notation above we let b_f (resp. b_s) be the size of B_f (resp. B_s).

We start by analyzing schedules for the fast machines. As no job in D and B_s can be assigned to fast machines, we focus only on the schedule of jobs in A, B_f and C. We call the jobs in A and B_f light jobs while we refer to the jobs in C as heavy jobs. We let $l = a + b_f$, i.e., l is the number of light jobs. Among those jobs we say that $l_1 \in \{0, \ldots, l\}$ are assigned to machine 1 (and thus $l - l_1$ is the number of light jobs on machine m). Similarly, we let $c_m \in \{1, 2\}$ be the number of heavy jobs assigned to machine m.[2] For each value of $b_f \in \{0, \ldots, b\}$ we want to minimize the following function:

$$\frac{(l_1 + 2 - c_m)(l_1 + (2 - c_m)t) + (l - l_1 + c_m)(l - l_1 + c_m t)}{r}. \tag{12}$$

Claim. For each value of $b_f \in \{0, \ldots, b\}$ the total latency (12) is minimized by setting $c_m = 2$ and $l_1 = l$.

[2] Observe that $c_m = 0$ corresponds (modulo renaming the machines) to the case in which $c_m = 2$. Indeed, machines 1 and m do have the same speed and thus we only need to consider schedules in which heavy jobs are either assigned to the same fast machine or to different fast machines.

Proof. To study the function above we consider two different cases: (i) $c_m = 2$ and (ii) $c_m = 1$.

Whenever $c_m = 2$ simple calculations show that (12) is minimized for $l_1 = (l + t + 1)/2$. But since $t \geq a + b \geq l$ we have that $l < (l + t + 1)/2$. This implies that, when $c_m = 2$, the minimum of (12) is achieved by setting $l = l_1$, implying the following total latency in this case:

$$\frac{l^2 + 4t}{r}.$$

Whenever $c_m = 1$ simple calculations show that (12) is minimized for $l_1 = l/2$. But l could not be even: indeed, although a is even b_f is not necessarily even. However, since $c_m = 1$, we have the same load on both machines 1 and m by assigning $\lfloor l/2 \rfloor$ to them. Thus, as fast machines have the same speed, we can assign the additional job either to 1 or m without changing the total latency of the solution. We then conclude that when $c_m = 1$, (12) is minimized for $l_1 = \lfloor l/2 \rfloor$, implying the following total latency in this case:

$$\frac{(\lfloor l/2 \rfloor + 1)(\lfloor l/2 \rfloor + t) + (\lceil l/2 \rceil + 1)(\lceil l/2 \rceil + t)}{r}.$$

We now conclude the analysis of the optimal schedule on the fast machines by arguing that:

$$l^2 + 4t \leq (\lfloor l/2 \rfloor + 1)(\lfloor l/2 \rfloor + t) + (\lceil l/2 \rceil + 1)(\lceil l/2 \rceil + t) = lt + l + 2t + \lfloor l/2 \rfloor^2 + \lceil l/2 \rceil^2 \tag{13}$$

thus implying that the best solution is achieved by setting $c_m = 2$ and $l_1 = l$. To show (13) we have two cases: when $l = 2$ (l cannot be less than 2), (13) is trivially true; otherwise we observe that it is equivalent to

$$t \geq \frac{l^2 - l - \lfloor l/2 \rfloor^2 - \lceil l/2 \rceil^2}{l - 2} \tag{14}$$

But

$$\frac{l^2 - l - \lfloor l/2 \rfloor^2 - \lceil l/2 \rceil^2}{l - 2} \leq l^2 \tag{15}$$

and since $t \geq l^2$ (14) is verified. □

Claim above shows that no matter how jobs of B are split in B_f and B_s the best solution is to schedule heavy jobs on one of the fast machine and light jobs on the other fast machine. Now, we conclude the proof by considering jobs B_s and D. None of these jobs can be scheduled on the fast machines. Since the schedule of jobs in D is fixed, we only need to argue what is the best size of B_s when jobs in A, B_f and C are optimally scheduled. Next claim shows that the total latency is minimized by setting $b_s = 0$.

Claim. Let S be any scheduling in which $c_m = 2$, $l_1 = l$ and $b_s > 0$. It holds $C(S) \geq C(OPT)$.

Proof. The proof of the claim is by induction on b_s.

We start by the base of the induction, i.e. $b_s = 0$. Since in this case $S = OPT$ the claim is true.

As for the inductive step we next show that if all solutions with $b_s = k$ satisfy the claim (true by inductive hypothesis) then so is for all solutions with $b_s = k{+}1$. Let S be one of such solutions, i.e., S has $b_s = k + 1$. Consider the solution S' obtained from S by moving a job in B from a machine $i \in \{2, \ldots, m-1\}$ to machine 1. Since S' has $b_s = k$ then by inductive hypothesis we have that $C(S') \geq C(OPT)$. Next we show that $C(S) \geq C(S')$ thus showing the claim. Let us denote by $\hat{n}_i$ the number of jobs of B on machine i in the schedule S. We have:

$$\begin{aligned} & C(S) - C(S') = \\ = & \tfrac{(a+b-(k+1))^2}{r} - \tfrac{(a+b-k)^2}{r} + (1+\hat{n}_i)\left(\tfrac{t}{r} + \hat{n}_i\right) - \hat{n}_i\left(\tfrac{t}{r} + \hat{n}_i - 1\right) = \\ = & \tfrac{t}{r} + 2\hat{n}_i - \tfrac{1}{r} - \tfrac{2(a+b-(k+1))}{r}. \end{aligned}$$

Since $r \geq 1$ and $n_i \geq 1$ from the choice of $t \geq 2(a+b) \geq 2(a+b-(k+1))$ the lemma follows. □

This concludes the proof of the lemma. □

In the theorem below we consider the following Nash schedule for this instance. There are $\lceil r \rceil w$ jobs of A on machine 1 and as many on machine m; there are w jobs of B on every machine $i = 2, \ldots, m-1$; a job of C is assigned to machine 1 and the other one to machine m, each job of D is assigned to its unique allowed machine. In this solution, machines 1 and m have both $\lceil r \rceil w + 1$ jobs and a total load of $\frac{t}{r} + \frac{\lceil r \rceil w}{r} \geq \frac{t}{r} + w$. Each slow machine has $w+1$ jobs and a total load of $\frac{t}{r} + w$. We next argue that this solution is a Nash equilibrium. Indeed, no job can unilaterally improve its latency moving to another machine. In details, each job of A suffers from a latency of $\frac{t}{r} + \frac{\lceil r \rceil w}{r}$ and moving to the other allowed machine would experience a latency of $\frac{t}{r} + \frac{\lceil r \rceil w}{r} + \frac{1}{r}$. Every job of B suffers from a latency of $\frac{t}{r} + w$ and moving to another machine i, with $i = 2, \ldots, m-1$, would suffer from a latency of $\frac{t}{r} + w + 1$, whereas moving on machine 1 or machine m would experience a latency greater or equal than $\frac{t}{r} + w + \frac{1}{r}$. Every job in C suffers from a latency of $\frac{t}{r} + \frac{\lceil r \rceil w}{r}$ and moving to another allowed machine would suffer from a latency of $2\frac{t}{r} + \frac{\lceil r \rceil w}{r}$. So, this scheduling is a Nash Equilibrium. Let us call it $NASH$.

Theorem 5. *With respect to Total Latency social function, job scheduling game with related machines has* $PoA = \Omega\left(\min\left(\frac{m+\sqrt{t}}{m+3}, \frac{n}{m+3}\right)\right)$.

Proof. We start by upper bounding the total latency of the optimal solution OPT. It holds:

$$\begin{aligned} C(OPT) &= 4\frac{t}{r} + (m-2)\frac{t}{r} + \frac{(a+b)^2}{r} \\ &\leq (m+3)\frac{t}{r} \end{aligned} \tag{16}$$

where the inequality is due to the definition of t.

We next lower bound the total latency of $NASH$.

$$\begin{aligned} C(NASH) &= 2(\lceil r \rceil w + 1)\left(\frac{t}{r} + \frac{\lceil r \rceil w}{r}\right) + (m-2)(w+1)\left(\frac{t}{r} + w\right) \\ &\geq 2rw\frac{t}{r} + 2\lceil r \rceil w^2 + mw\frac{t}{r} + m\frac{t}{r} + mw^2 + mw - 2w\frac{t}{r} - 2w^2 \\ &\geq \left[\frac{(k-1)(m+3)+3}{2r+m-2}(2r+m-2) + m\right]\frac{t}{r} + (2\lceil r \rceil + m - 2)w^2 \\ &\geq k(m+3)\frac{t}{r}. \end{aligned} \tag{17}$$

By (16) and (17) we have

$$PoA = \frac{C(NASH)}{C(OPT)} \geq k.$$

To completes the proof we need to show the greatest value that k could assume. If m and t are given (so n is univocally defined), then k has to satisfy the following condition:

$$t \geq (|A| + |B|)^2 \tag{18}$$

$$|A| \geq 1 \tag{19}$$

From the first condition follows that $k \leq \frac{m+\sqrt{t}}{m+3}$.

If, instead, m and n are given (and thus t is univocally determined), then the condition (18) is replaced by

$$|A| + |B| + |C| + |D| = n$$

that, implies $k \leq \frac{n}{m+3}$.

Putting together these conditions, we obtain the claimed bound. □