Software ENgineering

SEN

Online Scheduling of Splittable Tasks in Peer-To-Peer Networks

L. Epstein, R. van Stee

REPORT SEN-E0404 APRIL 21, 2004

ISSN 1386-369X

Online Scheduling of Splittable Tasks in Peer-To-Peer Networks

ABSTRACT
We consider online scheduling of splittable tasks on parallel machines. In our model, each task can be split into a limited number of parts, that can then be scheduled independently. We consider both the case where the machines are identical and the case where some subset of the machines have a (fixed) higher speed than the others. We design a class of algorithms which allows us to give tight bounds for a large class of cases where tasks may be split into relatively many parts. For identical machines we also improve upon the natural greedy algorithm in other classes of cases.

2000 Mathematics Subject Classification: 68W25, 68W40
1998 ACM Computing Classification System: F.2.2
Keywords and Phrases: online scheduling; splittable tasks; competitive ratio; greedy algorithm

Online Scheduling of Splittable Tasks in Peer-To-Peer Networks

Leah Epstein[*]   Rob van Stee[†]

Abstract

We consider online scheduling of splittable tasks on parallel machines. In our model, each task can be split into a limited number of parts, that can then be scheduled independently. We consider both the case where the machines are identical and the case where some subset of the machines have a (fixed) higher speed than the others. We design a class of algorithms which allows us to give tight bounds for a large class of cases where tasks may be split into relatively many parts. For identical machines we also improve upon the natural greedy algorithm in other classes of cases.

1 Introduction

In this paper, we consider the problem of distributing tasks on parallel machines, where tasks can be split in a limited amount of parts. A possible application of the splittable tasks problem exists in peer-to-peer networks [5]. In such networks large files are typically split and the parts are downloaded simultaneously from different locations, which improves the quality of service (QoS). More generally, computer systems often distribute computation between several processors. This allows the distributed system to speed up the execution of tasks. Naively it should seem that the fastest way to run a process would be to let all processors participate in the execution of a single process. However in practice this is impossible. Set-up costs and communication delays limit the amount of parallelism possible. Moreover, some processes may have limited parallelism by nature. In many cases, the best that can be done is that a process may be decomposed into a limited number of pieces each of which must be run independently on a single machine.

The definition of the model is as follows. In the sequel, we call the tasks "jobs" as is done in the standard terminology. We consider online scheduling of splittable jobs on m parallel machines. A sequence of jobs is to be scheduled on a set of machines. Unlike the basic model which assumes that each job can be executed on one machine (chosen by the algorithm), for splittable jobs, the required processing time p_j of a job j may be split in an arbitrary way into (at most) a given number of parts ℓ. Those parts become independent and may run in parallel or at different times on different processors. After a decision (on the way a job is split) has been made, the scheduler is confronted by the basic scheduling problem, where each piece of job is to be assigned non-preemptively to one machine. In the on-line version, jobs are presented to the algorithm in a list, this means that each job must be assigned before the next job is revealed. Only after the process of job splitting and assignment is completed, the next job is presented to the algorithm. The goal is to minimize the makespan which is the last completion time of any part of job.

We consider two machine models. The first one is the well known model of identical machines, where all machines have the same speed (w.l.o.g. speed 1). The second case relates to systems where several processors are faster (by some multiplicative factor) than the others. In this case let s be the speed of the

[*]School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. lea@idc.ac.il. Research supported by Israel Science Foundation (grant no. 250/01).

[†]Centre for Mathematics and Computer Science (CWI), Amsterdam, The Netherlands. Rob.van.Stee@cwi.nl. Research supported by the Netherlands Organization for Scientifi c Research (NWO), project number SION 612-061-000.

fast processors. The other processors have speed 1. This also contains the model where one processor is fast and all others are identical [15, 8, 3, 14]. We call the machines of speed s *fast*, and all other machines are *regular* machines. The number of fast machines is denoted by f whereas the number of regular machines is $m - f$. The processing time of job j on a machine of speed s is p_j/s. Each machine can process only one job (or part of job) at a time, and therefore the completion time of the machine is the total processing time of all jobs assigned to it (normalized by the speed), which is also called the *load* of the machine. In the context of downloading files in a peer-to-peer network, the speeds correspond to the bandwidths for the different connections.

We use competitive analysis and given a problem we would like to determine its competitive ratio. The competitive ratio of an algorithm is the worst case ratio between the makespan of the schedule produced by the algorithm, and the makespan of an optimal offline algorithm which receives all input tasks as a set and not one by one. We denote the cost of this optimal offline algorithm by OPT. The competitive ratio of a problem is the best possible competitive ratio that can be achieved by a deterministic on-line algorithm.

Previous work: The basic model (with $\ell = s = 1$) was studied in a sequence of papers, each improving either the upper bound or the lower bound on the competitive ratio [10, 7, 2, 12, 1, 9, 6, 11]. The offline splittable jobs problem was studied by Shachnai and Tamir [19]. They showed that the problem is NP-hard (already for identical machines) and gave a PTAS for uniformly related machines. The problem was also studied by Krysta, Sanders and Vöcking [13] who gave an exact algorithm which has polynomial running time for any constant number of uniformly related machines. A different model that is related to our model is scheduling of parallel jobs. In this case, a job has several identical parts that must run simultaneously on a given number of processors [4, 16].

Our results: We first analyze a simple greedy-type algorithm that splits jobs into at most ℓ parts, while assigning them in a way that the resulting makespan is as small as possible. We then introduce a type of algorithm that always maintains a subset of $k < \ell$ machines with maximal load (while maintaining a given competitive ratio), and show that it is optimal as long as ℓ is sufficiently large in relation to $m+f$. The case $f = m - 1$ is treated separately. For smaller ℓ, we give an algorithm for identical machines that uniformly improves upon our greedy algorithm. Finally, we consider the special case of four identical machines and $\ell = 2$, which is the smallest case for which we did not find an optimal solution. The algorithms assume that it is always possible to compute the value of OPT for a subsequence of jobs which already arrived. In section 3 we explain how to compute this value.

2 A greedy algorithm

In this section, we analyze a simple greedy-type algorithm that works as follows. Recall that we consider the case where there is a group of f machines of speed $s \geq 1$, and the remaining $m - f$ machines have speed 1. For each arriving job, the algorithm finds the way to schedule it on at most ℓ machines, in a way that the resulting makespan is as small as possible. This is done by assigning the job to a subset of least loaded fast machines and a subset of least loaded regular machines. To implement this algorithm, we need to consider the combination of the least loaded x regular machines with the least loaded y fast machines, for all feasible cases: $x + y \leq \ell$, $0 \leq x \leq \min\{\ell, m - f\}$ and $0 \leq y \leq \min\{\ell, f\}$. There are only $O(\ell^2)$ such combinations. If the job is split into less than ℓ parts, it means that the makespan did not change. Note that for $\ell = s = 1$, this algorithm reduces to the standard greedy algorithm for load balancing.

Consider an arbitrary subset S of ℓ machines, and denote the number of fast machines in this subset by g. Consider the time where the maximum load is achieved first. This happened after assigning a job on **exactly** ℓ machines. Denote the total processing time scheduled on the i-th machine in subset S by W_i^S ($i = 1, \dots, \ell$). Let x be the job that achieves the maximum load (and by a slight abuse of notation, also its processing time is denoted by x). Let $W = \sum_{i=1}^{m} W_i$, i.e. the total processing time of all jobs right before the assignment of x. Let GREEDY denote the makespan of the greedy algorithm. By our assignment, we have for any subset S

$$\text{GREEDY} \le \frac{W_1^S + \ldots + W_\ell^S + x}{sg + \ell - g} \Rightarrow (sg + \ell - g)\text{GREEDY} \le W_1^S + \ldots + W_\ell^S + x.$$

There are $\binom{m}{\ell}$ such subsets, and each machine occurs in $\binom{m-1}{\ell-1}$ of them. Summing the above inequality over all $\binom{m}{\ell}$ subsets, we have that each time a fast machine occurs, it contributes s to the left hand side; a regular machine contributes 1. Thus $\text{GREEDY} \cdot \left(s\binom{m-1}{\ell-1}f + \binom{m-1}{\ell-1}(m-f)\right) \le \binom{m-1}{\ell-1}(W_1 + \ldots W_m) + \binom{m}{\ell}x$ or $(sf + m - f)\text{GREEDY} \le W + x\frac{m}{l}$. Furthermore, we have $\text{OPT} \ge \frac{W+x}{sf+m-f}$. If $f \ge \ell$ we also have $\text{OPT} \ge \frac{x}{s\ell}$, otherwise $\text{OPT} \ge \frac{x}{sf+\ell-f}$. Thus if $f \ge \ell$

$$\text{GREEDY} \le \frac{W + x\frac{m}{\ell}}{sf + m - f} \le \text{OPT} + \frac{s\ell \cdot \text{OPT}(\frac{m}{\ell} - 1)}{sf + m - f} \le \left(1 + \frac{m - \ell}{sf + m - f} \cdot s\right)\text{OPT}.$$

and otherwise

$$\text{GREEDY} \le \text{OPT} + \frac{(sf + \ell - f)\text{OPT}(\frac{m}{\ell} - 1)}{sf + m - f} = \left(1 + \frac{sf + \ell - f}{sf + m - f}\left(\frac{m}{\ell} - 1\right)\right)\text{OPT}.$$

These ratios are decreasing in ℓ and are 1 for $\ell = m$. For $f = 0$ (or equivalently $s = 1$) the second ratio applies, which then becomes $2 - \ell/m$. For larger f, the ratio is lower.

3 Computing OPT

Throughout the paper we assume that the value of OPT is known to the on-line algorithm. There are several options to achieve this knowledge. The algorithm of Krysta, Sanders and Vöcking [13] can solve an offline problem exactly using time which is polynomial seeing the number of machines as constant. The drawback is that their algorithm must be exercised after every arrival of a job to find out the new value of OPT. Another and better option is simply to use the two following lower bounds on OPT: the sum of processing times of all jobs divided by the sum of speeds, and the size of the largest job divided by the sum of speeds of the ℓ fastest machines. We already used these bounds in Section 2.

All the proofs of upper bounds use only these bounds on OPT, and therefore the knowledge of the actual values of OPT is not required. Naturally, those bounds are not always tight as the offline problem is NP-complete already for identical machines and any constant ℓ [19]. Note that in almost all cases in this paper where we got tight bounds on the competitive ratio, the value of OPT is actually given by the maximum of the two bounds on OPT. This is always true for $\ell \ge (m+1)/2$. In these cases an optimal offline schedule (not only its cost) can be computed by the following algorithm. This algorithm works for the general case of uniformly related machines (where each machine i has some speed s_i).

Algorithm Calculate the value of OPT. We say that a job *fits* on a subset of machines if it can be placed there without any machine exceeding a load of OPT (normalized by the speed). Sort the machines by nondecreasing speeds.

Consider the largest job J. Clearly it fits on the ℓ fastest machines. Let i be an index such that J fits on machines $i, \ldots, i + \ell - 1$, where all these machines except possibly the last are used completely. If there is such an i, assign J there. We are left with machines $1, \ldots, i - 1, i + \ell, \ldots, m$ and possibly a part of machine $i+\ell-1$. This is a subset of at most ℓ machines, since $\ell \geq (m+1)/2$. Hence the remaining jobs can be split perfectly among these machines. Since the other machines are filled completely, they must all fit.

If there is no such index i, then J fits on machines $1,\ldots,\ell-1$ or less machines (note that these are the slowest machines). This implies that all jobs fit on at most ℓ machines: we need to add 1 to the number of machines used for one job since for later jobs we get that both the first machine of the job and the last one can be occupied partially by other jobs. Hence all jobs can be assigned without wasting any space.

4 Algorithm HIGH$(k,\mathcal{R})$

An important algorithm that we work with is the following, called HIGH(k,R). It maintains the invariant that there are at least k *regular* machines with load exactly $\mathcal{R}$ times the optimal load, where $\mathcal{R}$ is the competitive ratio that we want to prove. Clearly such an invariant can only be maintained for k at most equal to $\ell-1$ (consider the assignment of the first job), and in certain cases k has to be chosen even lower than that to get the best ratio. We will use this algorithm in the context of identical machines and in the case where there are several fast machines of speed s. Recall that the identical machines case is a special case of the second case (with $s=1$). We immediately present the more general algorithm.

On arrival of a job J of size x, HIGH$(k,\mathcal{R})$ assigns the job to at most ℓ machines such that the invariant is kept. We denote the optimal makespan before the arrival of J by OPT_1, and after the arrival of J by OPT_2. We would like to sort the machines by the capacity of jobs they can accommodate. For a machine i, let L_i be its load and s' be its speed ($s'=1$ or $s'=s$). Let b_i be the *gap* on machine i, which is the maximum load that can be placed on the machine in this step. That is, $b_i = s'(\mathcal{R}\cdot\text{OPT}_2 - L_i)$ for $i=1,\ldots,m$. We first sort only the regular machines in non-increasing order by their gaps. Clearly, the machines which had load $\mathcal{R}\text{OPT}_1$ have the smallest gap. We get $b_1 \geq \ldots \geq b_{m-f}$ and $b_{m-f-k+1} = \ldots = b_{m-f} = \mathcal{R}\text{OPT}_2 - \mathcal{R}\text{OPT}_1$.

Let $S_i = b_i + \ldots + b_{i+k-1}$ for $1 \leq i \leq m-f-k+1$. This is the sum of the gaps on k consecutive regular machines. The algorithm can work only under the condition that $S_{m-f-k+1} \leq x$: if x is smaller, then after assigning x there are less than k machines with load $\mathcal{R}\text{OPT}_2$. We distinguish two cases.

Case 1: $S_1 \geq x$. We can find a value i such that $S_i \geq x$ and $S_{i+1} \leq x$. If $S_i = x$, we can clearly assign J such that there are k regular machines with load $\mathcal{R}\text{OPT}_2$.

Suppose $S_i > x$. Then $i \leq m-f-k$ since $S_{m-f-k+1} \leq x$. We use the machines $i,\ldots,i+k$. This is a set of $k+1$ machines. We add b_j to machine j for $j=i+1,\ldots,i+k$ and put the nonzero remainder on machine i. The remainder fits there since the job can fit on machines $i,\ldots,i+k-1$ even without machine $i+k$. Clearly we get at least k regular machines with load $\mathcal{R}\text{OPT}_2$. The assignment is feasible since $\ell \geq k+1$.

Case 2: $S_1 < x$. Here we introduce another condition which is the following. Consider the k regular machines with the largest gaps, and among the machines that are not the k regular machines with smallest gap, choose another $\ell-k$ machines with largest gaps. The condition for the algorithm to succeed is that the sum of these ℓ gaps is at least the size x. The assignment of x first fills the gaps on the k least loaded regular machines, and the non-zero remainder is spread between the $\ell-k$ machines with largest gaps.

We use this algorithm several times in this paper. Each time, to show that it maintains some competitive ratio $\mathcal{R}$, we will show the following two properties.

1. A new job is never too large to be placed as described. That is, if we place it on the ℓ machines, k of which are the regular machines with largest gaps, and the other $\ell-k$ are the machines with the largest

gaps among the others (excluding the regular machines that have maximum load before), then afterwards the load on these machines is at most $\mathcal{R}\text{OPT}_2$.

2. A new job is never too small for the invariant to be maintained. I.e. if we assign the job on the k machines that had load $\mathcal{R}\text{OPT}_1$, then it fits exactly in the gaps, or there is a remainder. This will show that in all cases we can make at least k machines have load $\mathcal{R}\text{OPT}_2$.

Note that for each arriving job, the new value of OPT can be computed in time $O(1)$, and the worst step in algorithm HIGH$(k, \mathcal{R})$ with regard to the time complexity is maintaining the sorted order of the regular machines, which can be done efficiently.

4.1 Many splits

We consider the case $\ell \geq (m+f)/2$ (since $k \leq \ell - 1$, if $f = 0$ we need $\ell \geq (m+1)/2$). Note that this leaves open the case of $f = m-1$. This case will be considered separately in the next subsection.

We need some definitions in order to state the next Lemma. Let ℓ' be the sum of speeds of the ℓ fastest machines and let m' be the sum of all speeds. Clearly $\ell \geq f$ and so $\ell' = sf + \ell - f$ and $m' = sf + m - f$. Let $c = \ell'/m'$ and

$$\mathcal{R}_1(c) = \frac{1}{c^2 - c + 1}.$$

Note that $\mathcal{R}_1(c) = \mathcal{R}_1(1-c)$. Finally, let c_1 be the real solution to $c^3 - c^2 + 2c - 1 = 0$ ($c \approx 0.56984$).

Lemma 1 *For $c \geq c_1$, algorithm* HIGH$(m-\ell, R_1(c))$ *maintains a competitive ratio of $\mathcal{R}_1(c)$.*

Proof Let $k = m - \ell \leq \ell - 1$. We first show that the new job is never too large to be placed as described. If it is put on the ℓ machines which are all machines that did not have maximum load before the arrival of J, then the other $k = m - \ell$ regular machines have load $\mathcal{R}_1(c)\text{OPT}_1$ because of the invariant (they were the machines with highest load). Thus we need to show that $\ell'\mathcal{R}_1(c)\text{OPT}_2 + k\mathcal{R}_1(c)\text{OPT}_1 \geq W + x$ where W is the total load of all the jobs before J arrived.

We have $\text{OPT}_1 \geq W/m'$, $\text{OPT}_2 \geq (W+x)/m'$ and $\text{OPT}_2 \geq x/\ell'$. Therefore

$$\text{OPT}_2 \geq \alpha\frac{W+x}{m'} + (1-\alpha)\frac{x}{\ell'} \qquad \text{for any } 0 \leq \alpha \leq 1 \tag{1}$$

Taking $\alpha = \ell'/m'$, we get $k\text{OPT}_1 + \ell'\text{OPT}_2 \geq kW/m' + \ell'\alpha(W+x)/m' + \ell'(1-\alpha)x/\ell' = (W + x)(\alpha\ell'/m' + 1 - \alpha) = (W+x)(1 - \ell'/m' + \ell'^2/m'^2) = \frac{W+x}{\mathcal{R}_1(c)}$, as needed.

Second, we show that J is always large enough such that we can again make k regular machines have load $\mathcal{R}_1(c)\text{OPT}_2$. That is, $x \geq k\mathcal{R}_1(c)(\text{OPT}_2 - \text{OPT}_1)$. There are three possibilities for OPT_2: it is either x/ℓ', $(W+x)/m'$ or y/ℓ', where y is the processing time of some old job.

If $\text{OPT}_2 = y/\ell'$ we are done, since then $\text{OPT}_1 = y/\ell'$ as well. Otherwise, we use that $\text{OPT}_1 \geq W/m'$. Thus $\text{OPT}_2 - \text{OPT}_1 \leq \max(x/\ell', x/m') = x/\ell'$. We need to show that $k\mathcal{R}_1(c)x/\ell' \leq x$ or $k\mathcal{R}_1(c) \leq \ell'$. This holds if $c^3 - c^2 + 2c - 1 \geq 0$, which holds for $c \geq c_1$. This completes the proof of the upper bound of HIGH$(m-\ell, \mathcal{R}_1(c))$. □

Lemma 2 *No algorithm for the scheduling of ℓ-splittable jobs on a system of f fast machines of speed s and $m-f$ regular machines has a better competitive ratio than $\mathcal{R}_1(c)$.*

Proof The values m' and ℓ' are defined as above. Thus $m' = sf + m - f$. Furthermore, ℓ' is the sum of speeds of the ℓ fastest machines, so $\ell' = sf + \ell - f$ if $\ell \geq f$, $\ell' = s\ell$ otherwise. The lower bound consists of very small jobs of total size $m' = sf + m - f$, followed by a single job of size $W - m'$, where W will be determined later. The optimal offline makespan after the small jobs is $\text{OPT}_1 = 1$, and after the large job it is $\text{OPT}_2 = W/m'$.

Consider an online algorithm $\mathcal{A}$. After the small jobs have arrived, the algorithm "knows" it has to keep room for another single job. Therefore it can load the $m-\ell$ machines it is not going to use for that job with the maximum load $\mathcal{R}\text{OPT}_1$ (if it puts more on some machine, the final job does not arrive). There are many cases according to how many fast machines it loads. Let k_1 be the number of fully loaded regular machines and $k_2 = m-\ell-k_1$ the number of fully loaded fast machines.

If $\mathcal{A}$ maintains a competitive ratio of $\mathcal{R}$, we must have that $W \leq \mathcal{R}\text{OPT}_1(k_1+sk_2)+\mathcal{R}\text{OPT}_2((m-f-k_1)+s(f-k_2))$. This implies

$$\mathcal{R} \geq \frac{W}{m-\ell-k_2+sk_2+\text{OPT}_2(k_2+\ell-f+sf-sk_2)}. \tag{2}$$

We can see that this number is minimized by minimizing k_2, since the coefficient of k_2 in the denominator is $(\text{OPT}_2-1)(1-s)<0$. Therefore the lower bound is obtained by taking $k_2=0$ if $\ell \geq f$, and $k_2 = f-\ell$ otherwise. We choose W such that $W-m' = m'\ell'/(m'-\ell')$. We rewrite (2) to get $W \leq (m'-\ell')\mathcal{R}\text{OPT}_1 + \ell'\mathcal{R}\text{OPT}_2$. Then since $\text{OPT}_1 = 1$ and since from $W=(m')^2/(m'-\ell')$ follows $\text{OPT}_2 = m'/(m'-\ell')$, we get $\mathcal{R} \geq \frac{(m')^2}{(m'-\ell')^2+m'\ell'} = \frac{(m')^2}{(m')^2-m'\ell'+(\ell')^2} = \mathcal{R}_1(c)$. $\square$

These two lemmas imply the following theorem.

Theorem 1 *For $\ell'/m' \geq c_1$ and $\ell \geq \frac{m}{2}+\frac{1}{2}\max(f,1)$ (i.e. $f \neq m-1$), the algorithm* $\text{HIGH}(m-\ell,\mathcal{R}_1(c))$ *is well-defined and optimal.*

4.2 The case of $f=m-1$ fast machines

For completeness, in this section we consider the case $f=m-1$. We give tight bounds for many cases, including the case of $m-1$ parts, i.e. each job may run on all machines but one. Clearly we already solved the cases $f=0,\ldots,m-2$ and $f=m$ (this is the same case as $f=0$) for large enough ℓ. The solution of the case $f=m-1$ is very different from the other cases. First the algorithm is not the same for all values of s. For small s, for the first time we use an invariant on the fast machines. For large s, for the first time we do not use all the machines. Again we use m' as the sum of all speeds, i.e. $m'=(m-1)s+1$, and ℓ' as the sum of speeds of the ℓ fastest machines, i.e. $\ell'=s\ell$. We introduce a new notation k' which is the sum of speeds of the machines that are kept at maximum load. This value is determined by the algorithm.

For large s, we use an algorithm which never uses the regular machine. For the case $\ell=m-1$ it is a simple greedy algorithm that splits each job in a way that it keeps the load balanced on all fast machines. This gives the algorithm the ratio $1+\frac{1}{s(m-1)}$ (easily proved by area considerations). For $\ell<m-1$ the algorithm ignores the regular machine, and uses $\text{HIGH}(m-1-\ell,\mathcal{R}_{21})$ on $m-1$ fast machines only, where $\mathcal{R}_{21}$ is defined as a function of m' and ℓ' (which are functions of m, ℓ and s):

$$\mathcal{R}_{21} = \frac{(m')^2}{(m')^2-(m'-\ell')(\ell'+1)} = \frac{(m')^2}{(m')^2-m'-k'\ell'} .$$

We have $k'=sk=s(m-\ell-1)$. The algorithm keeps $k=m-\ell-1$ fast machines with load $\mathcal{R}_{21}\text{OPT}$. Since k must be smaller than ℓ, we require $\ell \geq m/2$.

On arrival of a job, let OPT_1 and OPT_2 be the optimal offline makespan before and after the arrival of the new job, respectively. The algorithm is the same as before but the properties are slightly different. We need to show that the following two properties hold:

1. $x \geq k'\mathcal{R}(\text{OPT}_2-\text{OPT}_1)$.
2. The gaps on the ℓ least loaded fast machines can contain x.

The second property can be reformulated as

$$\ell'\mathcal{R}\text{OPT}_2 + k'\mathcal{R}\text{OPT}_1 \geq W + x$$

where W is the total processing time of jobs which arrived before the job of processing time x. This follows from $\ell + k = m - 1$. Regarding the first property, similarly to before, we can bound the difference of the optimal offline costs by $\text{OPT}_2 - \text{OPT}_1 \leq x/\ell'$. This gives the condition $\mathcal{R}_{21} \leq \ell'/k'$.

To show the second property we again use the bounds $\text{OPT}_1 \geq \frac{W}{m'}$ and (1). We need to show

$$\frac{k'W}{m'} + \ell'\left(\alpha\frac{W+x}{m'} + (1-\alpha)\frac{x}{\ell'}\right) \geq \frac{W+x}{\mathcal{R}}.$$

Taking $1 - \alpha = \frac{k'}{m'}$, we get that this condition is satisfied for $\mathcal{R} = R_{21}$.

For small s, we use a variation on previous algorithms. The algorithm keeps $k = m - \ell$ *fast* machines with load $\mathcal{R}\text{OPT}$, where

$$\mathcal{R}_{22} = \frac{m'^2}{m'^2 - (m' + s - 1 - \ell')\ell'} = \frac{(m')^2}{(m')^2 - k'\ell'}. \tag{3}$$

The value we use for k' is $k' = s(m - l)$. The algorithm is defined as $\text{HIGH}(m - \ell, \mathcal{R}_{22})$, except that the roles of the fast machines and the regular machine have been reversed. In other words, we use the gaps on *fast* machines to fit the job, and if it needs more room we use at most $m - k - 1$ fast machines and the regular machine as well.

On arrival of a job, let OPT_1 and OPT_2 be the optimal offline makespan before and after the arrival of the new job, respectively. We again need the following two properties to hold:

1. $x \geq k'\mathcal{R}(\text{OPT}_2 - \text{OPT}_1)$.
2. The gaps on the $m - k$ other machines (that do not maintain the invariant) can contain x.

$$(m' - k')\mathcal{R}\text{OPT}_2 + k'\mathcal{R}\text{OPT}_1 \geq W + x.$$

The first property again translates into $\mathcal{R}_{22} \leq \ell'/k'$. To show the second property we again use the bounds $\text{OPT}_1 \geq \frac{W}{m'}$ and (1). We need to show

$$\frac{k'W}{m'} + (m' - k')\left(\alpha\frac{W+x}{m'} + (1-\alpha)\frac{x}{\ell'}\right) \geq \frac{W+x}{\mathcal{R}}.$$

Taking $1 - \alpha = \frac{k'\ell'}{m'(m'-k')}$, we get that this condition is satisfied for $\mathcal{R} = R_{22}$.

We now give a lower bound that proves that these bounds are tight. The lower bound is actually more general, and holds for all values of ℓ and s.

Lemma 3 *For $f = m - 1$, any online algorithm has a competitive ratio of at least* $\min(\mathcal{R}_{21}, \mathcal{R}_{22})$.

Proof We define a sequence of jobs with the following processing times: $P_1 = 1,\ P_j = \frac{\ell'}{m'-\ell'}\sum_{i=1}^{j-1} P_i$. Let OPT_j be the optimal offline cost on the subsequence of the first j jobs. Then we see that for $j \geq 3$ we have

$$\text{OPT}_j = \frac{1}{m' - \ell'}\sum_{i=1}^{j-1} P_i = \frac{P_j}{\ell'} \quad \text{and} \quad P_j = \frac{m'}{m' - \ell'}P_{j-1}.$$

Consider the behavior of the on-line algorithm starting from the **third job**.

If the algorithm never splits a job using the regular machine, we need to consider two cases. If $\ell = m - 1$, the competitive ratio tends to the ratio $1 + \frac{1}{s(m-1)}$ of the greedy algorithm that does not use the regular machine. The second case $\ell \leq m-2$ is slightly more difficult. Only the first two jobs might be scheduled on the regular machine. Consider job P_j. If $\mathcal{A}$ maintains a competitive ratio of $\mathcal{R}$ until this point, then on each of the fast machines that it does not use for job j it has placed at most $sR\text{OPT}_{j-1}$, and we find

$$\frac{\sum_{i=3}^{j} P_i - (m-\ell-1)sR\text{OPT}_{j-1}}{\ell'} \leq \mathcal{R}\text{OPT}_j$$

which implies that $\mathcal{R}(\ell'\text{OPT}_j + s(m-\ell-1)\text{OPT}_{j-1}) + P_1 + P_2 \geq \sum_{i=1}^{j} P_i$. We use $\sum_{i=1}^{j} P_i = P_j + \sum_{i=1}^{j-1} P_i = P_j(1+\frac{m'-\ell'}{\ell'}) = \frac{m'}{\ell'}P_j$ to rewrite this condition in terms of P_j, and divide by P_j. For large enough j we can neglect P_1 and P_2 and find

$$\mathcal{R}\left(1+\frac{s(m-\ell-1)(m'-\ell')}{m'\ell'}\right) \geq \frac{m'}{\ell'}.$$

This gives $\mathcal{R} \geq \mathcal{R}_{21}$.

Otherwise (some job uses the regular machine), let j be the index of the first job for which a part is assigned to the regular machine. If $\mathcal{A}$ maintains a competitive ratio of $\mathcal{R}$ until this point, then on the machines that it does not use for job j (which are all fast) it has placed at most $sR\text{OPT}_{j-1}$, and we find

$$\frac{\sum_{i=1}^{j} P_i - s(m-\ell)R\text{OPT}_{j-1}}{s(\ell-1)+1} \leq \mathcal{R}\text{OPT}_j$$

which implies that $\mathcal{R}(\text{OPT}_j(s(\ell-1)+1) + s(m-\ell)\text{OPT}_{j-1}) \geq \sum_{i=1}^{j} P_i$. We use $\sum_{i=1}^{j} P_i = \frac{m'}{\ell'}P_j$ to rewrite this condition in terms of P_j, and divide by P_j to find

$$\mathcal{R}\left(\frac{s\ell-s+1}{\ell'}+\frac{s(m-\ell)(m'-\ell')}{\ell'm'}\right) \geq \frac{m'}{\ell'}$$

which leads to $\mathcal{R} \geq \mathcal{R}_{22}$. $\square$

We summarize our results in the following Theorem.

Let $s_1 = (m-1+\sqrt{m^2-2m+1+4\ell})/(2\ell)$.

Theorem 2 *For the case of $m-1$ fast machines of speed s. If $s \geq s_1$, and ($m/2 \leq \ell \leq m-2$ and $\mathcal{R}_{21} \leq \ell'/(m'-\ell'-1)$) or $\ell = m-1$, then the optimal competitive ratio of any online algorithm is $\mathcal{R}_{21}$. If $s \leq s_1$, $\ell > m/2$ and $\mathcal{R}_{22} \leq \ell'/(m'-\ell'+s-1)$, then the optimal competitive ratio of any online algorithm is $\mathcal{R}_{22}$.*

Corollary 1 *For $f = \ell = m-1$, the optimal competitive ratio is $\min(R_{21}, \mathcal{R}_{22})$.*

Proof For small s, if $\ell = m-1$ then the value of $\mathcal{R}_{21}$ is defined properly to be $1+\frac{1}{s(m-1)}$, attained by the greedy algorithm that only uses fast machines. This ratio is thus tight.

For large s, if $\ell = m-1$ then the first property to be checked leads to the condition $s\mathcal{R}(\text{OPT}_2 - \text{OPT}_1) \leq x$. Similarly to before, we can bound the difference of the optimal offline costs by $\text{OPT}_2 - \text{OPT}_1 \leq x/(sm-s)$. Using (3), this leads to the condition $s^2(m-1)^2 \leq (m-2)(sm-s+1)^2$. This is true since $s(m-1) < sm-s+1$ and $m \geq 3$. Thus the condition on the ratio in Theorem 2 is satisfied as well as the condition on ℓ. $\square$

4.3 Few splits on identical machines

Following Theorem 1, we now consider the case $c < c_1$. Let

$$\mathcal{R}_3(c) = \frac{1}{2}\left(c^2 - c + 2 - (c-1)\sqrt{c^2+4}\right).$$

We examine algorithm $\textsc{High}(\ell/\mathcal{R}_3(c), \mathcal{R}_3(c))$, i.e. $k = \ell/\mathcal{R}_3(c)$, and verify that it maintains a competitive ratio of $\mathcal{R}_3(c)$. The second condition is immediately satisfied, since the only relevant case is $\textsc{opt}_2 - \textsc{opt}_1 \le x/\ell$, which leads to the constraint $kR_3(c) \le \ell$ as in the previous subsection. Moreover, we have that $k + \ell \le m$ for all $c \le c_1$, since $c/R_3(c) + c \le 1$ for $c < c_1$.

Suppose a new job is placed on the ℓ machines with lowest load. By the invariant and since $k+\ell \le m$, there are k machines with load $\mathcal{R}_3(c)\textsc{opt}_1$. Denote the total load on the remaining machines (not the k old machines or the ℓ machines that were just used) by V. Then

$$V \ge (W - k\mathcal{R}_3(c)\textsc{opt}_1) \cdot \frac{m-k-\ell}{m-k}$$

since these machines were not the least loaded machines before the new job arrived.

Thus we need to check that

$$k\mathcal{R}_3(c) \cdot \textsc{opt}_1 + \ell\mathcal{R}_3(c) \cdot \textsc{opt}_2 + V \ge W + x$$

or

$$k\mathcal{R}_3(c) \cdot \textsc{opt}_1 \cdot \frac{\ell}{m-k} + \ell\mathcal{R}_3(c) \cdot \textsc{opt}_2 \ge W \cdot \frac{\ell}{m-k} + x.$$

As before, we use that $\textsc{opt}_1 \ge W/m$ and $\textsc{opt}_2 \ge \alpha\frac{W+x}{m} + (1-\alpha)\frac{x}{\ell}$ for any $0 \le \alpha \le 1$. We take $\alpha = \frac{m-k}{2m-k-\ell} \le \frac{m-k}{m} \in [0,1]$.

We find

$$\frac{k\ell\textsc{opt}_1}{m-k} + \ell\textsc{opt}_2 \ge \left(\frac{k\ell}{m-k} + \ell\alpha\right)\frac{W}{m} + \left(\ell \cdot \frac{\alpha}{m} + 1 - \alpha\right)x \ge \frac{\frac{W\ell}{m-k} + x}{\mathcal{R}_3(c)},$$

since $\mathcal{R}_3(c)$ satisfies $\mathcal{R}_3(c) = \frac{2m-k-cm}{m-kc}$ (using $k = \ell/\mathcal{R}_3(c) = cm/\mathcal{R}_3(c)$).

Theorem 3 *For $\ell/m < c_1$, the algorithm $\textsc{High}(\ell/\mathcal{R}_3(c), \mathcal{R}_3(c))$ maintains a competitive ratio of $\mathcal{R}_3(c)$, where $c = \ell/m$.*

We now show a lower bound for this case. This lower bound uses a technique originally introduced by Sgall [17, 18].

Theorem 4 *For m divisible by ℓ, the competitive ratio of any randomized (or deterministic) algorithm is at least $\frac{1}{1-\left(1-\frac{\ell}{m}\right)^{m/\ell}}$. This gives a general lower bound of $\mathcal{R}_4(c) = \left(1 - \left(\frac{c-1}{c}\right)^c\right)^{-1}$ for $c = \ell/m$.*

Proof Fix a sequence of random bits to be used by the algorithm. Start with $(m-\ell)/\ell$ jobs of size ℓ. Then define $\mu = m/(m-\ell)$ and give jobs J_i of size $\ell\mu^{i-1}$ for $i = 1, \ldots, m/\ell$.

Since $\mu - 1 = \ell/(m-\ell)$, we have $\sum_{i=1}^{m/\ell} \ell\mu^{i-1} = \ell\frac{\mu^{m/\ell}-1}{\mu-1} = (m-\ell)(\mu^{m/\ell}-1)$. Therefore the total size of all the jobs is $W = m - \ell + (m-\ell)(\mu^{m/\ell}-1) = (m-\ell)\mu^{m/\ell} = m\mu^{m/\ell-1}$. After job J_i has arrived we have $\textsc{opt}_i = \mu^{i-1}$. So $\sum_{i=1}^{m/\ell} \textsc{opt}_i = (\mu^{m/\ell}-1)/(\mu-1)$.

For $1 \le i \le m$, let L_i be the load of machine i at the end of the sequence after sorting the machines by non-increasing load. Removing any $i-1$ jobs still leaves a machine with load of at least $L_{\ell i+1}$. Therefore $\mathcal{A}(J_m) = L_1$, $\mathcal{A}(J_{m-1}) \ge L_{\ell+1}$ and in general $\mathcal{A}(J_i) \ge L_{\ell(m-i)+1} \ge \frac{1}{\ell}\sum_{j=1}^{\ell} L_{\ell(m-i)+j}$, so $\sum \mathcal{A}(J_i) \ge W/\ell$.

It follows that

$$\mathcal{R} \ge \frac{W/\ell}{\sum_{i=1}^{m/\ell} \textsc{opt}_i} \ge \frac{m\mu^{m/\ell-1}(\mu-1)/\ell}{\mu^{m/\ell}-1} = \frac{\mu^{m/\ell}}{\mu^{m/\ell}-1} = \frac{1}{1-\left(\frac{m}{m-\ell}\right)^{-m/\ell}} = \frac{1}{1-\left(1-\frac{\ell}{m}\right)^{m/\ell}}.$$

The value of the lower bound tends to $e/(e-1)$ for $m/\ell \to \infty$, for instance when ℓ is constant and m grows without bound. For $m = c\ell$ we find a lower bound of

$$\mathcal{R}_4(c) = \left(1 - \left(\frac{c-1}{c}\right)^c\right)^{-1},$$

independent of m. □

We give an overview of the various upper and lower bounds in Figure 1.

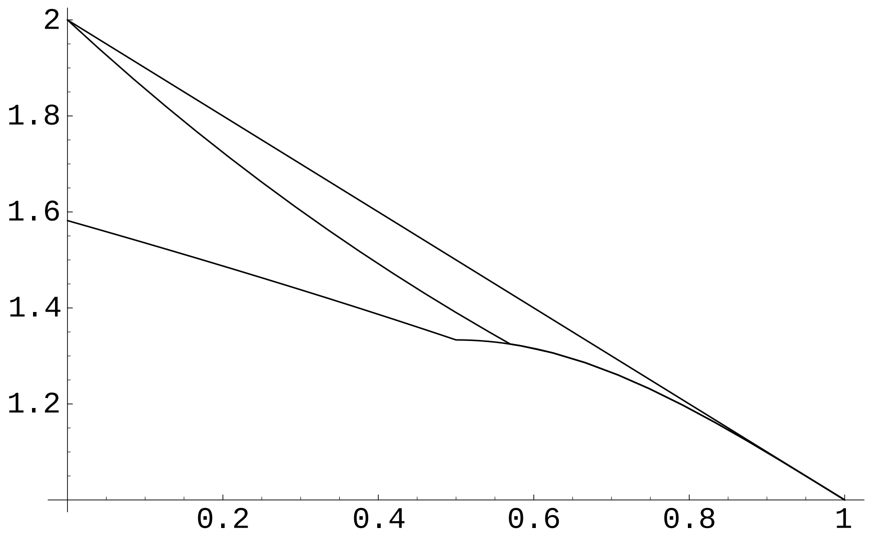

Figure 1: Upper and lower bounds for identical machines. The horizontal axis is ℓ/m, the vertical axis is the competitive ratio. The top line is the greedy algorithm, the middle line is our best upper bound and the lower line is our best lower bound. For $c \leq 1/2$, this lower bound also holds for randomized algorithms.

5 A special case: four machines, two parts

Already for this sub-problem it is nontrivial to give an optimal algorithm. Surprisingly, in this case the lower bound from Lemma 2 is not tight. This hints that for the cases where we do not give matching upper bounds, it is likely that the lower bounds are simply not the best possible.

For the case of three parts the previous section gives an algorithm of competitive ratio $16/13 \approx 1.23$. For two parts, we use the algorithm $\textsc{High}(1, 10/7)$ which maintains the invariant that at least one machine has load exactly $\frac{10}{7}\textsc{opt}$. Note that our greedy algorithm maintains only a competitive ratio of $1 + \frac{\ell}{m}(\frac{m}{\ell} - 1) = 3/2$.

Theorem 5 *For four machines and 2-splittable jobs, the algorithm* $\textsc{High}(1, 10/7)$ *maintains a competitive ratio of* $10/7$.

Proof The proof proceeds similarly to before.

First, we show that a new job is not too large. Suppose it is placed on the two lowest machines. Then the other machines have load $\frac{10}{7}\textsc{opt}_1$ (because of the invariant) and $\beta \geq (W - \frac{10}{7}\textsc{opt}_1)/3$ (because it was the second highest machine before J arrived). The total load on all the machines must be bounded by $\frac{10}{7}\textsc{opt}_1 + \frac{20}{7}\textsc{opt}_2 + \beta \geq \frac{10}{7}(\frac{2}{3}\textsc{opt}_1 + 2\textsc{opt}_2) + W/3$.

Similarly to before, we find

$$\frac{\text{OPT}_1}{3} + \text{OPT}_2 \geq \frac{W}{12} + \frac{\alpha W}{4} + \frac{\alpha x}{4} + (1-\alpha)\frac{x}{2} = \frac{7}{60}(2W+3x)$$

by taking $\alpha = 3/5$. Therefore $\frac{10}{7}\text{OPT}_1 + \frac{20}{7}\text{OPT}_2 + \beta \geq W + x$, as needed.

Second, we show that a new job is always large enough so that the new maximum load is $10/7$ times the optimal load. We have $\text{OPT}_2 - \text{OPT}_1 \leq x/2$, and $\frac{10}{7}\frac{x}{2} \leq x$. □

Lemma 4 *Any on-line algorithm for minimizing the makespan of 2-splittable jobs on four parallel machines has a competitive ratio of at least* $\mathcal{R}_4 = (47 - \sqrt{129})/26 \approx 1.37085$.

Proof Suppose $\mathcal{A}$ maintains a competitive ratio of R. Two jobs of size 2 arrive. $\text{OPT} = 1$ (already after the first job). We number the machines from 1 to 4, and denote the loads of the machines by $M_1 \geq M_2 \geq M_3 \geq M_4$. If $\mathcal{A}$ puts the first two jobs on two or fewer machines, we are done immediately. This leaves us with two cases. We use $\mathcal{A}$ to also denote the makespan of $\mathcal{A}$.

Case 1. $\mathcal{A}$ puts the first two jobs on 3 machines. Then $M_4 = 0$, $M_1 \leq \mathcal{R}_4$, $M_3 \geq 4 - 2\mathcal{R}_4$, $M_2 + M_3 \geq 4 - \mathcal{R}_4$ and therefore $M_2 \geq (4 - \mathcal{R}_4)/2 = 2 - \mathcal{R}_4/2$.

A job x of size 2 arrives. If $\mathcal{A}$ puts no part of x on machine 4, we are done since $M_3 + 1 \geq 5 - 2\mathcal{R}_4 > 3\mathcal{R}_4/2$ (we have $\text{OPT} = 3/2$).

So $\mathcal{A}$ must put a part of x on machine 4. Finally, a job of size 6 will arrive. The best thing $\mathcal{A}$ can do is to put it on the two machines with lowest load (after x has been assigned). Which machines are these?

Case	Lowest load is on	and is at least
1a	2 and 3, 1 and 3 or 1 and 2	$4 - \mathcal{R}_4$
1b	2 and 4	$8 - 4\mathcal{R}_4$
1c	3 and 4	$8 - 4\mathcal{R}_4$

This covers the cases, since if part of x is put on 4, either machine 2 or machine 3 receives nothing and remains lower than machine 1. We now prove the entries in the last column.

Case 1a. Suppose machines 2 and 3 are the lowest. Already before assigning x we had $M_2 + M_3 \geq 4 - \mathcal{R}_4$. Now suppose machines 1 and 3 are the lowest. Clearly $M_1 + M_3 \geq M_2 + M_3 \geq 4 - \mathcal{R}_4$. Finally, if machines 1 and 2 are the lowest then $M_1 + M_2 \geq M_3 + M_2 \geq 4 - \mathcal{R}_4$.

Case 1b. It must be that x goes to machines 3 and 4. $\mathcal{A}$ should put as little as possible on 4 in order to minimize the load on the two lowest machines after this (2 and 4). $\mathcal{A}$ can put at most $3\mathcal{R}_4/2 - (4 - 2\mathcal{R}_4) = 7\mathcal{R}_4/2 - 4$ on machine 3 and thus puts at least $2 - (7\mathcal{R}_4/2 - 4) = 6 - 7\mathcal{R}_4/2$ on machine 4. After this, the load of the two lowest machines (2 and 4) is at least $2 - \mathcal{R}_4/2 + 6 - 7\mathcal{R}_4/2 = 8 - 4\mathcal{R}_4$.

Case 1c. Again $\mathcal{A}$ should put as little as possible on 4 in order to minimize the load on the two lowest machines after this (3 and 4). It can put at most $3\mathcal{R}_4/2 - (2 - \mathcal{R}_4/2) = 2\mathcal{R}_4 - 2$ on machine 1 or 2 and must therefore put at least $2 - (2\mathcal{R}_4 - 2) = 4 - 2\mathcal{R}_4$ on machine 4. After this, the load of the two lowest machines (3 and 4) is at least $8 - 4\mathcal{R}_4$.

This concludes the discussion of the subcases. We find that after assigning x, the load on the two lowest machines is at least $\min(4 - \mathcal{R}_4, 8 - 4\mathcal{R}_4) = 8 - 4\mathcal{R}_4$ since $\mathcal{R}_4 > 4/3$. Finally the job of size 6 arrives, now $\text{OPT} = 3$ and $\mathcal{A} \geq (8 - 4\mathcal{R}_4 + 6)/2 > 3\mathcal{R}_4$.

Case 2. $\mathcal{A}$ puts the first two jobs on 4 machines, each machine has one part of one job. Then $M_2 + M_3 = M_1 + M_4 = 2$ and

$$M_1 \leq \mathcal{R}_4.$$

It is possible that a job of size 4 arrives. Then $\text{OPT} = 2$ and $\mathcal{A}$ must be able to place it such that $\mathcal{A} \leq 2\mathcal{R}_4$. Therefore we must have $(M_3 + M_4 + 4)/2 \leq 2\mathcal{R}_4$ or

$$M_3 + M_4 \leq 4(\mathcal{R}_4 - 1).$$

Together these equations give

$$M_1 + M_2 \geq 8 - 4\mathcal{R}_4, \qquad M_2 \geq 8 - 5\mathcal{R}_4 \qquad \text{and} \qquad M_4 = 2 - M_1 \geq 2 - \mathcal{R}_4.$$

Thus, if these inequalities do not hold after the first two jobs arrive, a job of size 4 arrives and we are done. Otherwise, we let a job of size x ($x \leq 1$) arrive where x will be determined later. Then $\text{OPT} = 1 + x/4$. After this a final job of size $y = x + 4$ will arrive. We have a similar division into cases as in Case 1.

Case	Lowest load is on	and is at least
2a	2 and 3, 1 and 3 or 1 and 2	2
2b	(1 or 2) and 4	$10 - 6\mathcal{R}_4$
2c	3 and 4	$10 - 6\mathcal{R}_4$

Case 2a. We have $M_2 + M_3 = 2$, the rest is as in Case 1a.

Case 2b. We have $M_2 + M_4 \geq 10 - 6\mathcal{R}_4$, so also $M_1 + M_4 \geq 10 - 6\mathcal{R}_4$.

Case 2c. We are left with the case where machines 3 and 4 are the lowest. We will choose x so large that it cannot be assigned to machines 1 and 2 only: $M_1 + M_2 + x > 2\mathcal{R}_4(1 + x/4)$, in other words $x > (12\mathcal{R}_4 - 16)/(2 - \mathcal{R}_4)$.

Thus some part of x is assigned to machine 3 or 4. $\mathcal{A}$ will use machines 3 and 4 for the last job, so it is best to put as much of x as possible on 1 or 2. WLOG this part is put on machine 2 since $M_2 \leq M_1$. Denote the part of x that is assigned to machine i by x_i. We have $x_2 \leq (1 + x/4)\mathcal{R}_4 - M_2$ and

$$M_3 + x_3 = 2 - M_2 + x - x_2 \geq 2 - M_2 + x(1 - \mathcal{R}_4/4) - \mathcal{R}_4 + M_2 = 2 - \mathcal{R}_4 + x(1 - \mathcal{R}_4/4).$$

Therefore $M_3 + M_4 + x_3 \geq 4 - 2\mathcal{R}_4 + x(1 - \mathcal{R}_4/4)$.

We take x such that $10 - 6\mathcal{R}_4 = 4 - 2\mathcal{R}_4 + x(1 - \mathcal{R}_4/4)$, in other words $x = (24 - 16\mathcal{R}_4)/(4 - \mathcal{R}_4) = (16\sqrt{129} - 128)/(\sqrt{129} + 57) \approx 0.7859$. Note that $x > (12\mathcal{R}_4 - 16)/(2 - \mathcal{R}_4)$, as needed.

This concludes the discussion of the subcases. We find that the load of the two lowest machines is at least $10 - 6\mathcal{R}_4$ after assigning job x, independently of $\mathcal{A}$'s decision. (Note $10 - 6\mathcal{R}_4 < 2$ for $\mathcal{R}_4 > 4/3$.)

After the last job arrives, $\text{OPT} = y/2$. The best thing that $\mathcal{A}$ can do is to put y on the two machines with lowest load. Its final load is thus at least $(10 - 6\mathcal{R}_4 + y)/2$. The competitive ratio is $(10 - 6\mathcal{R}_4 + 4 + x)/(4 + x) = \mathcal{R}_4$. □

6 Conclusion

This paper considered the classical load balancing model in the context of parallelizable tasks. We designed and analyzed several algorithms, and showed tight bounds for many cases. As for open problems, there is a large amount of work done on various multiple machines scheduling and load balancing problems. Many of those on-line (and offline) problems are of interest to be studied for scenarios where parallelization is allowed.

For the special case of four machines and two parts, which is the smallest case for which we do not have a tight solution, we show a lower bound of 1.37085 and an upper bound of $10/7$. This is a better lower bound than Lemma 2, hinting that in areas where our bounds are not tight, the lower bound can be improved.

References

[1] Suzanne Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29:459–473, 1999.

[2] Yair Bartal, Howard Karloff, and Yuval Rabani. A better lower bound for on-line scheduling. *Information Processing Letters*, 50:113–116, 1994.

[3] Yookun Cho and Sartaj Sahni. Bounds for List Schedules on Uniform Processors. *SIAM Journal on Computing*, 9(1):91–103, 1980.

[4] Anja Feldmann, Jiří Sgall, and Shang-Hua Teng. Dynamic scheduling on parallel machines. *Theoretical Computer Science*, 130:49–72, 1994.

[5] Amos Fiat and Jared Saia. Censorship resistant Peer-to-Peer content addressable networks. In *Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Algorithms (SODA 2002)*, pages 94–103, 2002.

[6] Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. *Journal of Scheduling*, 3:343–353, 2000.

[7] Gabor Galambos and Gerhard J. Woeginger. An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling. *SIAM Journal on Computing*, 22:349–355, 1993.

[8] Teofilo F. Gonzalez, Oscar H. Ibarra, and S. Sahni. Bounds for LPT Schedules on Uniform Processors. *SIAM Journal on Computing*, 6(1):155–166, 1977.

[9] Todd Gormley, Nick Reingold, Eric Torng, and Jeffery Westbrook. Generating adversaries for request-answer games. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 564–565, 2000.

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

[11] J.F. Rudin III. *Improved bounds for the on-line scheduling problem*. PhD thesis, The University of Texas at Dallas, May 2001.

[12] David R. Karger, Steven J. Phillips, and Eric Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20:400–430, 1996.

[13] Piotr Krysta, Peter Sanders, and Berthold Vöcking. Scheduling and traffic allocation for tasks with bounded splittability. In *Proc. of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS 2003)*, pages 500–510, 2003.

[14] Rongheng Li and Lijie Shi. An on-line algorithm for some uniform processor scheduling. *SIAM Journal on Computing*, 27(2):414–422, 1998.

[15] Jane W. S. Liu and C. L. Liu. Bounds on scheduling algorithms for heterogeneous computing systems. In Jack L. Rosenfeld, editor, *Proceedings of IFIP Congress 74*, volume 74 of *Information Processing*, pages 349–353, 1974.

[16] Edwin Naroska and Uwe Schwiegelshohn. On an on-line scheduling problem for parallel jobs. *Information Processing Letters*, 81(6):297–304, 2002.

[17] Jiří Sgall. A Lower Bound for Randomized On-Line Multiprocessor Scheduling. *Inf. Process. Lett.*, 63(1):51–55, 1997.

[18] Jiří Sgall. *On-Line scheduling on parallel machines*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, USA, 1994.

[19] Hadas Shachnai and Tami Tamir. Multiprocessor scheduling with machine allotment and parallelism constraints. *Algorithmica*, 32(4):651–678, 2002.