

## Multiprocessor Scheduling with Machine Allotment and Parallelism Constraints

H. Shachnai<sup>1</sup> and T. Tamir<sup>1</sup>

**Abstract.** Modern computer systems distribute computation among several machines to speed up the execution of programs. Yet, setup and communication costs, as well as parallelism constraints, bound the number of machines that can share the execution of a given application, and the number of machines by which it can be processed *simultaneously*. We study the resulting scheduling problem, stated as follows. Given a set of  $n$  jobs and  $m$  uniform machines, assign the jobs to the machines subject to parallelism and machine allotment constraints, such that the overall completion time of the schedule (or *makespan*) is minimized. Indeed, the *multiprocessor scheduling problem* (where each job can be processed by a *single* machine) is a special case of our problem; thus, our problem is strongly NP-hard.

We present a  $(1 + \alpha)$ -approximation algorithm for this problem, where  $\alpha \in (0, 1]$  depends on the minimal number of machine allotments and the minimal parallelism allowed for any job. Also, we show that when the maximal number of machines that can share the execution of a job is some fixed constant, our problem has a *polynomial time approximation scheme*; for other special cases we give optimal polynomial time algorithms. Finally, through the relation of our problem to the classic *preemptive* scheduling problem on multiple machines, we shed some fresh light on what is known in scheduling folklore as the *power of preemption*.

**Key Words.** Multiprocessor scheduling, Makespan, Machine allotments, Parallelizable jobs.

**1. Introduction.** A continuing trend in modern computer systems is to distribute computation among several physical processors. This enables us to speed up the execution of heavy applications. Ideally, the work required by such applications could be shared by *any* number of processors. However, setup and communication costs and the maximal level of parallelism within each application, bound the number of machines to which it can be *allotted*, and the number of machines by which it can be processed *simultaneously*.

The resulting scheduling problem can be stated as follows. Suppose that  $n$  jobs need to be scheduled on  $m$  machines; each machine,  $M_i$ ,  $1 \leq i \leq m$ , runs at specific rate,  $u_i$ ; each job,  $J_j$ ,  $1 \leq j \leq n$ , is associated with a processing time,  $t_j$ , an *allotment parameter*,  $a_j$ , and a *parallelism parameter*,  $\rho_j$ . Thus, the execution of  $J_j$  can be shared by at most  $a_j$  machines, and at most  $\rho_j$  machines can process  $J_j$  *simultaneously*. Our objective is to schedule the jobs on the machines, such that the allotment and parallelism constraints are satisfied, and the overall completion time of all jobs (or the *makespan*) is minimized. We call this problem *scheduling with parallelism and machine allotment constraints (SPAC)*. Indeed, the *multiprocessor scheduling problem*,<sup>2</sup> where each job can run on a

---

<sup>1</sup> Department of Computer Science, The Technion, Haifa 32000, Israel. {hadas,tami}@cs.technion.ac.il.

<sup>2</sup> Throughout the paper, sometimes we call this problem *non-preemptive scheduling*.

single machine (namely,  $\rho_j = a_j = 1, \forall 1 \leq j \leq n$ ), is a special case of our problem; thus, our problem is strongly NP-hard.<sup>3</sup>

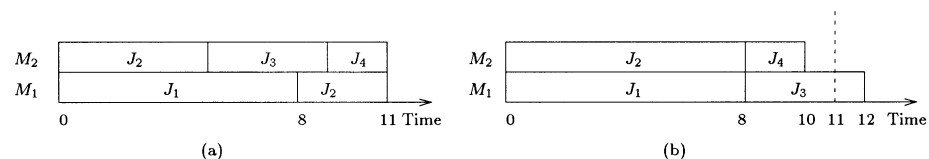
Note that the allotment parameter,  $a_j$ , bounds also the number of machines that can process  $J_j$  simultaneously. Thus, without loss of generality, we assume that  $\forall j, \rho_j \leq a_j$ . We also study in this paper the special case of SPAC in which  $\forall j, \rho_j = a_j$  (i.e., parallelism constraints do not affect the schedule). We refer to this case as the problem of *scheduling with machine allotment constraints (SAC)*.

Denote by  $w_{\text{OPT}}(I)$  the length of an optimal schedule of an instance  $I$ . Note that  $w^* = \sum_j t_j / \sum_i u_i$  is a lower bound for  $w_{\text{OPT}}(I)$ . When each job can be allotted (possibly, to run in parallel) to *any* number of machines, this lower bound is obtained by a simple greedy algorithm, based on McNaughton's rule [14]: it starts by scheduling the first job on the first machine; then it proceeds to the next job (whenever the current job,  $J_j$ , was allocated  $t_j$  processing units), or to the next machine (when the current machine runs for  $w^*$  time units). The resulting schedule incurs at most  $n + m - 1$  allotments of jobs to machines. However, we cannot predict how the machines will share the execution of the jobs. Thus, even if  $\sum_j a_j \geq n + m - 1$ , we may not be able to obtain the lower bound.

The next example shows how allotment/parallelism constraints come into play in finding a schedule which minimizes the makespan.

**EXAMPLE 1.1.** Consider a system with two identical machines  $M_1, M_2$ , whose rates are  $u_1 = u_2 = 1$ , and four jobs with  $t_1 = t_2 = 8, t_3 = 4, t_4 = 2$ , and  $\rho_j = 1, \forall 1 \leq j \leq 4$ . Figure 1(a) presents the schedule produced by a greedy algorithm. The makespan of this schedule is  $\sum_j t_j / \sum_i u_i = \frac{22}{2} = 11$ . Assume now that  $a_1 = a_2 = a_3 = 1$  and  $a_4 = 2$ . Note that the overall number of allotments allowed in the system is equal to the number of allotments incurred by the greedy algorithm; however, since only the execution of  $J_4$  can be shared by two machines, the best possible schedule has the length 12 (Figure 1(b)).

Note that the problem of *preemptive* scheduling on parallel machines can also be described as a special case of SPAC (take  $\rho_j = 1$  and  $a_j = m, \forall 1 \leq j \leq n$ ); while non-preemptive scheduling is strongly NP-hard, an optimal preemptive schedule can be found in polynomial time. We address here a natural extension of preemptive scheduling, in which we bound the number of machines that can share the execution of each job. Thus, for each job  $J_j, \rho_j = 1$  and  $1 < a_j < m$ . To the best of our knowledge, this problem is studied here for the first time. Though our answer is partial (namely, we consider only instances in which either the jobs or the machines are *identical*), our results show that, in fact, the presence of machine allotment constraints alone can distinguish between instances which are solvable in polynomial time, and instances which are strongly NP-hard.



**Fig. 1.** Scheduling with parallelism and machine allotment constraints.

<sup>3</sup> Generally, in SPAC we allow preemptions while processing a job,  $J_j$ , on some machine. However, when  $\forall j, \rho_j = a_j = 1$ , such preemptions are redundant.

We study our scheduling problem in both *uniform* and *identical* machine environments. For many fundamental scheduling problems, similar solvability/approximability results were obtained in these two environments (e.g., the makespan problem is optimally solvable in both environments when preemptions are allowed [14], [13], [9] and has a *polynomial time approximation scheme (PTAS)* in both environments when preemptions are not allowed [11], [12], [6]). Interestingly, our study shows a clear distinction between these two environments, with respect to the solvability of the SPAC problem (see Section 5).

**1.1. Motivation.** As mentioned earlier, the SPAC problem has important application in scheduling on multiprocessors, and in distributed computing. In distributed systems, load balancing and computation speedup are achieved by partitioning large applications to run on several machines; this is done through process migration (see, e.g., [21]). Therefore, the resources required by each job,  $J_j$ , are classified as either (i) *machine dependent* (e.g., peripheral devices), which can be allocated to  $J_j$  once it is scheduled to run on a *specific* machine, or (ii) *machine independent* (e.g., shared data), which can migrate with segments of the job from one machine to another, along its execution. The limited amount of machine-dependent resources sets a bound,  $a_j$ , on the number of machines that can share the execution of  $J_j$  (i.e., the number of machines to which  $J_j$  can be allotted throughout the schedule); the limited amount of machine-independent resources sets a bound,  $\rho_j$ , on the number of machines that can run  $J_j$  in *parallel*.

Another application is production planning. Production processes [5] typically involve the usage of consumable resources (i.e., special materials) which cannot migrate from one machine to another, and mobile resources (e.g., human supervision), which allow flexibility in the choice of machines. The maximal amount of consumable resources determines the allotment parameter of a production process; the available amount of mobile resources determines its parallelism parameter.

**1.2. Related Work.** The problem of scheduling a set of jobs on parallel machines with the objective of minimizing the makespan has been studied extensively (comprehensive surveys appear, e.g., in [10] and [3]). The non-preemptive scheduling problem is known to be strongly NP-hard [7], and admits a PTAS: the papers [11] and [1] give PTASs for identical machines, and [12] and [6] give PTASs for uniform machines. When preemptions are allowed, the makespan problem can be solved optimally in polynomial time. A greedy algorithm (McNaughton's rule [14]) is suitable for identical machines. For uniform machines, the first optimal algorithm was presented in [13]; an optimal algorithm which also minimizes the number of preemptions is given in [9]. When the allotment or parallelism parameter of a job is greater than one, the job can be *parallelized* to run simultaneously on several machines. Previous work on parallelizable jobs (see, e.g., [2] and [20]) assume that the processing of a job  $J_j$  is partitioned *evenly* among the machines which process this job. In contrast, in SPAC we allow *any* partition of the processing of  $J_j$  among several machines (as long as parallelism constraints are not violated). Recently, it was shown in [19] that for instances where each job can be allotted to any number of machines (i.e.,  $1 \leq \rho_j \leq m$  and  $a_j = m, \forall j$ ), the SPAC problem can be solved optimally.

Other related works deal with a special case of the *class-constrained multiple knapsack (CCMK)* problem [17], [18], in which a set of unit-sized items of  $m$  different types ( $u_i$

items of type  $i$ ) need to be placed in  $n$  bins; each bin has a limited capacity,  $t_j$ , and a bound,  $a_j$ , on the number of distinct types of items it can hold. The objective is to pack as many items as possible in the bins. The application of this problem to data placement on parallel disks was studied in [18] and [8]. When each knapsack is represented by a job with length  $t_j$  and allotment parameter  $a_j$ , and the items of type  $i$  are represented by a machine,  $M_i$ , with rate  $u_i$ , we get an instance of the SAC problem. Thus, some of the results in [18] and [8] can be adapted to special cases of the SAC problem. In particular, when the jobs are identical, and  $\sum_j a_j \geq m + n - 1$ , we get from [18] that the SAC problem can be solved optimally. The results in [8] imply that the SAC problem is strongly NP-hard when  $\sum_j a_j = m$  (i.e., each machine processes one job segment on the average), even when all the jobs are identical; for identical jobs, the *maximal utilization* problem (in which we wish to maximize the number of processing units completed within a given time interval) admits a PTAS.

**1.3. Our Results.** We describe below our main results. Unless specified otherwise, all of our results hold for *uniform* machines. Note that although in the SPAC problem we allow preemptions while processing a job on some machine, the algorithms presented in this paper do not use such preemptions.

In Section 2 we study the complexity of the SPAC problem. In particular, we show that SPAC is already strongly NP-hard for instances with no parallelism constraints (i.e., the SAC problem) and *weak* allotment constraints. Specifically, SAC is strongly NP-hard in the following cases:

1. On identical machines, where each job can be allotted to at least  $c$  machines, for any fixed  $c > 1$
2. On identical machines, where the *total* number of allotments is unbounded.
3. For identical jobs, where the total number of allotments is at least  $\frac{3}{2}m$ .

These hardness results extend the hardness result in [8], which holds for identical jobs with  $\sum_j a_j = m$ .

In Section 3 we present a  $\max_j(1 + 1/\rho_j)$ -approximation algorithm for the SPAC problem. Our algorithm proceeds in two steps: (i) Finding an infeasible schedule of optimal length, where a job  $J_j$  may be processed by  $\rho_j + 1$  processors. (ii) Transforming this schedule into a feasible one. The running time of the algorithm is  $O(\max(n \lg n, m \lg m))$ . This algorithm improves and generalizes an algorithm presented in [18] for the CCMK problem. For identical machines we modify this algorithm to obtain a  $\max_j(1 + 1/(2\rho_j - 1))$ -approximation ratio.

In Section 4 we give a PTAS for the SAC problem. Our PTAS can be used for instances in which the maximal allotment parameter of any job is some fixed constant. First, we show that the problem is strongly NP-hard in this case, even when the machines or the jobs are *identical*. Then we develop a PTAS which is based on the observation that the makespan may be extended by at most a factor of  $1 + \varepsilon$  if (i) small jobs are allotted to a single machine, and (ii) large jobs can be partitioned only to processing segments of certain lengths.

Section 5 explores the relation between the solvability of the classic preemptive scheduling problem, and the amount of machine allotments/parallelism allowed in the

schedule. We discuss instances in which  $\rho_j < a_j, \forall j$ . We call such instances *parallel-dominated*.

We show that the SPAC problem is strongly NP-hard for these instances, even when the jobs are identical and  $\forall j, a_j = \rho_j + 1$ , and solvable by an  $O(m \lg m)$  algorithm, when the jobs are identical and  $\forall j, a_j > \rho_j + 1$ . This implies, for example, that the preemptive scheduling problem (in which  $\rho_j = 1$ , for all  $1 \leq j \leq n$ ) becomes strongly NP-hard, when the execution of each job can be shared by at most *two* machines. Our results in Section 5 yield an interesting distinction between the solvability of our problem in the uniform and the identical machines environments. In particular, SPAC can be solved optimally in polynomial time for *parallel dominated* instances when the machines are identical; however, as mentioned above, on uniform machines the problem is strongly NP-hard.

**2. Hardness of the SAC Problem.** In this section we derive hardness results for the SAC problem. The case where  $\forall j, a_j = 1$  is known to be strongly NP-hard. When  $\forall j, a_j > 1$ , some preemptions are allowed for each of the jobs, and we may expect that the problem becomes easy to solve (as the classic makespan problem with preemptions). We show that the SAC problem is strongly NP-hard even for instances with no parallelism constraints, “weak” allotment constraints, and with identical machines/jobs. In other words, unless  $P = NP$ , it cannot already admit a *fully polynomial time approximation scheme (FPTAS)* for these instances.

We consider three classes of instances of the SAC problem. For each, we explain why it may seem to be “easy-to-solve” and follow this with a proof of hardness. The three classes are:

1. Identical machines, where each job can be allotted to at least  $c$  machines, for any  $c > 1$ .
2. Identical machines, where the *total* number of allotments is unbounded.
3. Identical jobs, where the total number of allotments is at least  $\frac{3}{2}m$ .

We derive our hardness results using reductions from *3-partition*, which is strongly NP-hard [7]. An instance of 3-partition is defined as follows.

*Input:* A finite set  $A$  of  $3q$  elements, a bound  $B \in \mathbb{Z}^+$ , and a size  $s(x)$  for each  $x \in A$ , such that each  $s(x)$  satisfies  $B/4 < s(x) < B/2$  and such that  $\sum_{x \in A} s(x) = qB$ .

*Output:* Is there a partition of  $A$  into  $q$  disjoint sets,  $S_1, S_2, \dots, S_q$ , such that, for  $1 \leq i \leq q$ ,  $\sum_{x \in S_i} s(x) = B$ ? (Note that the above constraints on the element sizes imply that every such  $S_i$  must contain exactly three elements from  $A$ .)

**2.1. Identical Machines and Any Number of Splits per Job.** We first consider instances with identical machines and any number of allotments per job. We show that there is no constant  $c$ , such that if each job could be allotted to  $c$  identical machines, then SAC admits an FPTAS.

**THEOREM 2.1.** *The SAC problem is strongly NP-hard, even if  $\forall 1 \leq j \leq n, a_j \geq c$ , for any given  $c > 1$  and the machines are identical.*

PROOF. Let  $c > 1$  be an integer. Given an instance of 3-partition, we construct an input,  $I$ , for the makespan problem with identical machines and  $\forall 1 \leq j \leq n, a_j \geq c$ , such that  $w_{\text{OPT}}(I) = 1$  if and only if  $A$  has a 3-partition.

The input for the makespan problem consists of  $m = (c - 1)3q + q$  machines with the same rates:  $u_1 = u_2 = \dots = u_{(c-1)3q+q} = B$ ; and  $n = 3q$  jobs with  $t_j = (c - 1)B + s(x_j)$ ,  $a_j = c$ ,  $\forall 1 \leq j \leq 3q$ . Thus, we have that  $\sum_j t_j = \sum_i u_i = (c - 1)3qB + qB$ . Since  $\sum_j t_j = \sum_i u_i$ ,  $w_{\text{OPT}}(I) \geq 1$ .

Assume that  $A$  has a 3-partition to the sets  $S_1, S_2, \dots, S_q$ . This induces the following schedule of  $I$ , whose makespan equals 1. For all  $1 \leq j \leq 3q$ ,  $J_j$  is processed for a single time unit on arbitrary vacant  $c - 1$  machines out of  $M_1, \dots, M_{(c-1)3q}$ . That is,  $(c - 1)B$  processing units are allocated to  $c - 1$  segments of  $J_j$ . In addition, if  $x_j \in S_k$  ( $k$  is the index of the triple to which  $x_j$  belongs in the partition), then the last ( $c$ )th segment of  $J_j$  is allocated  $s(x_j)$  processing units on  $M_{(c-1)3q+k}$ .

Thus, to each job we allocate exactly  $t_j = (c - 1)B + s(x_j)$  processing units on  $c$  different machines. Since  $\forall 1 \leq k \leq q, \sum_{j \in S_k} s(x_j) = B$ , each of the last  $q$  machines allocates exactly  $u_i = B$  processing units, and the makespan equals 1.

Assume that a schedule whose makespan equals 1 exists for  $I$ . We show that  $A$  has a 3-partition. For each machine,  $M_i$ , let  $n_i$  denote the number of jobs scheduled on  $M_i$ , and let  $J_{i_1}, J_{i_2}, \dots, J_{i_{n_i}}$  be the list of these jobs, such that, without loss of generality,  $i_1 < i_2 < \dots < i_{n_i}$ . The following graph,  $G = (V, E)$ , is induced by the schedule:

$V$ : there is a vertex,  $J_j$ , for each job,  $1 \leq j \leq 3q$ .

$E$ : each machine  $M_i$  contributes to  $E$  the edges of the path  $J_{i_1}, J_{i_2}, \dots, J_{i_{n_i}}$ .

Note that each machine contributes exactly  $n_i - 1$  edges to  $E$ . Therefore, the graph  $G$  has  $\sum_{i=1}^m n_i - m$  edges. Recall that  $J_j$  can be executed by at most  $a_j$  machines. In other words,  $J_j$  can appear on at most  $a_j$  paths, meaning that  $\sum_i n_i \leq \sum_j a_j$ . Therefore, the number of edges in  $G$  is at most  $\sum_j a_j - m = 3qc - (c - 1)3q - q = 2q$ . Having  $n = 3q$  vertices and at most  $2q$  edges,  $G$  consists of at least  $q$  connected components.

Assume that  $G$  has  $\ell$  connected components:  $D_1, D_2, \dots, D_\ell$ . Consider a component  $D_k = (V_{D_k}, E_{D_k})$ .  $V_{D_k}$  is a set of jobs.  $D_k$  is connected, therefore, for each machine,  $M_i$ , the path contributed by  $M_i$  is either completely contained or not contained in  $E_{D_k}$ . Thus,  $E_{D_k}$  determines the subset of machines which process the jobs in  $V_{D_k}$ .

CLAIM 2.2. *For each component  $D_k, \forall 1 \leq k \leq \ell$ , there exists an integer  $r_k > 0$  such that  $\sum_{j \in V_{D_k}} t_j = ((c - 1)|V_{D_k}| + r_k)B$ .*

PROOF. Recall that  $\sum_j t_j = \sum_i u_i$ ; thus, in any schedule with makespan equals 1, no machine is idle. It means that *all* the processing units of  $E_{D_k}$  are allocated to  $V_{D_k}$ . The rate of each machine is  $B$ . Thus, the total number of processing units allocated to the jobs in  $V_{D_k}$  is a multiple of  $B$ . Since  $\forall j, t_j > (c - 1)B$ , there exists an integer  $r_k > 0$  such that  $\sum_{j \in V_{D_k}} t_j = ((c - 1)|V_{D_k}| + r_k)B$ .  $\square$

We now prove that  $\ell = q$ , that is,  $G$  consists of exactly  $q$  connected components.

CLAIM 2.3. *The graph  $G$  has exactly  $q$  connected components.*

PROOF. From Claim 2.2 there exist positive integers  $r_1, \dots, r_\ell$  such that  $((c-1)|V_{D_1}| + r_1)B + \dots + ((c-1)|V_{D_\ell}| + r_\ell)B = ((c-1)3q + q)B$ . There are  $3q$  jobs, therefore,  $\sum_{k=1}^{\ell} |V_{D_k}| = 3q$ , and we get that  $\sum_{i=1}^{\ell} r_k = q$ . Since the  $r_k$ 's are positive integers and  $\ell \geq q$  we must have  $\ell = q$  and  $r_1 = \dots = r_q = 1$ .  $\square$

Now, given that  $G$  consists of  $D_1 \cup D_2 \cup \dots \cup D_q$ , define the following partition: for all  $1 \leq j \leq 3q$ ,  $j \in S_k$  if and only if  $j \in V_{D_k}$ . By Claim 2.3,  $r_k = 1, \forall 1 \leq k \leq q$ , thus,  $\sum_{j \in V_{D_k}} t_j = ((c-1)|V_{D_k}| + 1)B$ , meaning that  $\sum_{j \in S_k} s(x_j) = B$  for all  $1 \leq k \leq q$ .  $\square$

**2.2. Identical Machines and Any Total Number of Splits.** For the preemptive scheduling problem on identical machines, an optimal schedule can be obtained using at most  $m-1$  preemptions (For example, the greedy algorithm “splits” only the last job on each machine.) It means that the total number of allotments of jobs to machines is at most  $n+m-1$ . In our second hardness result we show that, for any given  $c$ , even if the machines are identical, and the *total* number of allotments may be larger than  $c(m+n)$ , the makespan problem is strongly NP-hard.

**THEOREM 2.4.** *The SAC problem is strongly NP-hard, even if  $\sum_{j=1}^n \min(m, a_j) \geq c(n+m)$ , and the machines are identical.<sup>4</sup> This holds for any  $c \geq 1$ .*

PROOF. For a given  $c$ , we show a reduction from the *3-partition problem*, with  $|A| = 3q \geq 15c$ . We construct an input,  $I$ , for the makespan problem with identical machines and  $\sum_j \min(m, a_j) \geq c(n+m)$ , such that  $w_{\text{OPT}}(I) = 1$  if and only if  $A$  has a 3-partition.

The input  $I$  consists of  $m = q$  machines with the same rates:  $u_1 = u_2 = \dots = u_m = q^2B + q$ ; and  $n = 4q$  jobs:

- $3q$  jobs with  $t_j = q^2s(x_j)$ ,  $a_j = 1, \forall 1 \leq j \leq 3q$ . We call these jobs *integral*.
- $q$  jobs with  $t_j = q$ ,  $a_j = q, \forall 3q < j \leq 4q$ . We call these jobs *additional*.

Note that  $\sum_i u_i = q^3B + q^2 = \sum_j t_j$ . In addition, for this instance,  $\sum_{j=1}^n a_j = q^2 + 3q$ ;  $m+n = q + 4q$ . Thus,  $\sum_j \min(m, a_j) = q^2 + 3q > c(5q)$ , for any  $q \geq 5c$ . Since  $\sum_j t_j = \sum_i u_i$ ,  $w_{\text{OPT}}(I) \geq 1$ .

We show that  $A$  has a 3-partition if and only if  $w_{\text{OPT}}(I) = 1$ . Assume that  $A$  has a 3-partition to the sets  $S_1, S_2, \dots, S_q$ . The following is a schedule whose makespan equals 1.

1. One processing unit of each machine  $M_i, 1 \leq i \leq m$ , is allocated to each of the additional jobs (i.e., overall  $M_i$  allocates  $q$  processing units). Thus, the execution of each additional job is shared among  $a_j = q$  machines.
2. The remaining  $q^2B$  processing units of  $M_i$  are allocated to the integral jobs  $\{J_{i_1}, J_{i_2}, J_{i_3}\}$  such that  $S_i = \{x_{i_1}, x_{i_2}, x_{i_3}\}$ .

Since  $\forall i, \sum_{j \in S_i} s(x_j) = B$ , the total number of processing units allocated by  $M_i$  in the second step is  $q^2B$ . Therefore, the above is a schedule whose makespan equals 1.

<sup>4</sup> We take the minimum between  $m$  and  $a_j$  since we gain nothing if a job can be allotted to more than  $m$  machines. This makes our result stronger.

Assume that there exists a schedule whose makespan equals 1 for  $I$ .

**CLAIM 2.5.** *In any such schedule, exactly  $q$  processing units of each machine are allocated to additional jobs.*

**PROOF.** Let  $n_i$  denote the number of processing units of  $M_i$  that are allocated to the additional jobs. The remaining  $u_i - n_i$  processing units are allocated to integral jobs. The integral jobs cannot split, thus, in any such schedule,  $u_i - n_i$  is a multiple of  $q^2$ . Since  $u_i = q^2B + q$  and the total processing time of the additional jobs is  $q^2$  we get that  $n_i = q$  for all  $i$ .  $\square$

Now, given that  $q$  processing units of each machine are allocated to additional jobs, we conclude that the remaining  $q^2B$  processing units of each machine are allocated to integral jobs, and a 3-partition of  $A$  is induced by the schedule.  $\square$

**2.3. Identical Jobs.** When  $\sum_j a_j = m$  and the jobs are identical, the SAC problem is strongly NP-hard: this can be shown by a simple reduction from 3-partition (as mentioned in [8]). We show that SAC remains strongly NP-hard, even if the jobs are identical, and the set of possible partitions is larger, more precisely,  $\sum_j a_j \geq \alpha m$ , for  $\alpha = \frac{3}{2}$ .

**THEOREM 2.6.** *The SAC problem is strongly NP-hard even if the jobs are identical and  $\sum_j a_j \geq \frac{3}{2}m$ .*

**PROOF.** Given an instance for 3-partition, we construct an input,  $I$ , for the makespan problem with  $\sum_j a_j \geq \frac{3}{2}m$ , such that  $w_{\text{OPT}}(I) = 1$  if and only if  $A$  has a 3-partition. In this reduction we adapt some ideas from the hardness proof given in [8].

The input  $I$  consists of  $m = 4q$  machines with the following rates: for the first  $3q$  machines,  $u_i = K - s(x_i)$ ,  $1 \leq i \leq 3q$ , where  $K > 3qB$  is a large constant; for the other  $q$  machines  $u_i = 3K + B$ ,  $3q < i \leq 4q$ . There are  $n = 3q$  identical jobs with  $t_j = 2K$ ,  $a_j = 2$ ,  $\forall 1 \leq j \leq 3q$ . For this instance,  $\sum_j t_j = \sum_i u_i = 6qK$ , and, as needed,  $\sum_j a_j = 6q = \frac{3}{2}m$ . Since  $\sum_j t_j = \sum_i u_i$ ,  $w_{\text{OPT}}(I) \geq 1$ .

We show that  $A$  has a 3-partition if and only if  $w_{\text{OPT}}(I) = 1$ . Assume that  $A$  has a 3-partition to the sets  $S_1, S_2, \dots, S_q$ . Let  $S_k = \{x_{k_1}, x_{k_2}, x_{k_3}\}$ ,  $\forall 1 \leq k \leq q$ . The following is a schedule whose makespan equals 1:  $\forall 1 \leq k \leq q$ , the four machines  $M_{k_1}, M_{k_2}, M_{k_3}, M_{3q+k}$  process the three jobs  $J_{k_1}, J_{k_2}, J_{k_3}$ . Specifically,  $M_{k_i}$  allocates  $K - s(x_{k_i})$  processing units to  $J_{k_i}$ ,  $1 \leq i \leq 3$ , and  $M_{3q+k}$  allocates  $K + s(x_{k_i})$  to  $J_{k_i}$ ,  $1 \leq i \leq 3$ . Since  $s(x_{k_1}) + s(x_{k_2}) + s(x_{k_3}) = B$ , and  $u_{3q+k} = 3K + B$ , the completion time equals 1. Note also that the allotment constraints are preserved: for any  $1 \leq j \leq 3q$ , if  $x_j \in S_k$ , then  $J_j$  is processed by the two machines  $M_j$  and  $M_{3q+k}$ .

Now, given a schedule whose makespan equals 1, we find a 3-partition of  $A$ . Denote by *slow* the set of the first  $3q$  machines  $\{M_1, \dots, M_{3q}\}$ . Note that for any pair,  $M_{i_1}, M_{i_2}$ , of slow machines, the total number of processing units provided by  $M_{i_1}$  and  $M_{i_2}$  is less than  $2K$ . Since  $t_j = 2K$ , if some  $J_j$  is scheduled only on these two machines it cannot be completed on time. Since  $a_j = 2$  for  $1 \leq j \leq n$ , this implies that each job,  $J_j$ , is processed by at most *one* slow machine. In addition, since there are  $3q = n$



slow machines, and no idle time is possible (since  $\sum_i u_i = \sum_j t_j$ ), each slow machine processes exactly one job. Assume, without loss of generality, that  $\forall 1 \leq i \leq 3q, M_i$  is the slow machine allocated to  $J_i$  (for the whole duration of the schedule). For each such job, since  $a_j = 2$ , the remaining  $K + s(x_i)$  processing units are allocated by a single machine. Consider a machine  $M_i, i > 3q$ . Since  $K$  was selected such that  $K \gg s(x)$ , for any  $x \in A$ , it follows that  $M_i$  processes exactly three jobs (since  $u_i = 3K + B$ , and the remainders of any four jobs require more than  $u_i$  processing units). Let  $J_{k_1}, J_{k_2}$ , and  $J_{k_3}$  be the three jobs scheduled on  $M_{3q+k}$ . We get that  $3K + B = 3K + (s(x_{k_1}) + s(x_{k_2}) + s(x_{k_3}))$ , meaning that  $x_{k_1}, x_{k_2}$ , and  $x_{k_3}$  form a triple for the partition. Since all the jobs are scheduled, the whole schedule of the jobs on the last  $q$  machines, induces a valid 3-partition of  $A$ .  $\square$

### 3. Approximation Algorithm for the SPAC Problem

3.1. *Uniform Machines.* In this section we present an approximation algorithm for the SPAC problem on uniform machines. Denote by  $w_{\text{OPT}}(I)$  the length of an optimal schedule of an instance  $I$ . First, we show that an optimal schedule can be obtained by relaxing the parallelism constraint<sup>5</sup> of each job by one. Specifically, if for all  $j$ , we run  $J_j$  in parallel on  $\rho_j + 1$  (instead of  $\rho_j$ ) machines, we can obtain a schedule of length  $w_{\text{OPT}}(I)$ . Then we transform the above infeasible optimal schedule into a feasible one, whose makespan is  $w \leq \max_j(1 + 1/\rho_j)w_{\text{OPT}}(I)$ .

#### 3.1.1. Relaxing the Parallelism Constraint

**THEOREM 3.1.** *Given an instance  $I$  of the SPAC problem, denote by  $I^+$  the instance in which the parallelism parameter of each job is increased by one (i.e.,  $\rho_j^+ = \rho_j + 1, \forall j$ ); then we can find in  $O(\max(m \lg m, n \lg n))$  steps a schedule for  $I^+$ , whose makespan is at most the minimal possible makespan for  $I$ .*

Given the instance  $I^+$  derived from  $I$ , we present a polynomial time algorithm which finds a legal schedule of  $I^+$ . The algorithm, denoted by  $\mathcal{A}_r$ , proceeds by scheduling  $J_j$  on at most  $\rho_j^+ = \rho_j + 1$  machines,  $\forall 1 \leq j \leq n$ . The length of the schedule generated for  $I^+$  by  $\mathcal{A}_r$  is at most the length of an optimal schedule of  $I$ .

We renumber the machines in nonincreasing order by their speeds, i.e.,  $u_1 \geq u_2 \geq \dots \geq u_m$ , and the jobs in a nonincreasing order by their *processing ratios*, i.e.,  $t_1/\rho_1 \geq t_2/\rho_2 \geq \dots \geq t_n/\rho_n$ . For each  $1 \leq \ell \leq n$ , let  $\hat{\rho}_\ell = \sum_{j=1}^\ell \rho_j$ . Let

$$(1) \quad w = \max \left\{ \frac{\sum_{j=1}^n t_j}{\sum_{i=1}^m u_i}, \max_{\{\ell | \hat{\rho}_\ell \leq m\}} \frac{\sum_{j=1}^\ell t_j}{\sum_{i=1}^{\hat{\rho}_\ell} u_i} \right\}.$$

To prove the theorem, we show that the makespan of any schedule of  $I$  is at least  $w$ , and that  $\mathcal{A}_r$  generates for  $I^+$  a schedule of length  $w$ . Note that, as illustrated in Example 1.1,  $w$  is not tight; that is, for some instances  $w_{\text{OPT}}(I) > w$ .

<sup>5</sup> When  $\rho_j = a_j$ , we relax at the same time the parallelism and the allotment constraint of  $J_j$ .

We first show that  $w$  is a lower bound on the length of any legal schedule of  $I$ .

LEMMA 3.2. *For any instance,  $I$ ,  $w_{\text{OPT}}(I) \geq w$ .*

PROOF. Consider a schedule of length  $w$ . When no machine is idle at any time during the schedule, the total processing potential of the machines is  $w \sum_i u_i$ . Thus,  $w_{\text{OPT}}(I)$  is at least the left term in the right-hand side of (1). The execution time of  $J_1$  is minimized when it runs in parallel on the  $\rho_1$  fastest machines for the whole duration of the schedule. Similarly, we cannot do better than scheduling the first  $\ell$  jobs on the  $\hat{\rho}_\ell$  fastest machines for the whole duration of the schedule. Thus, the right term in (1) is a lower bound for  $w_{\text{OPT}}(I)$ .  $\square$

We now turn to describing and analyzing the algorithm  $\mathcal{A}_r$ .  $\mathcal{A}_r$  adapts some ideas from the approximation algorithm presented in [18] for the CCMK problem. In each stage we represent by  $Q_i$  the *potential* of the machine  $M_i$ , that is, the number of processing units that  $M_i$  can still allocate. Initially,  $Q_i = wu_i$ .  $\mathcal{A}_r$  maintains a list,  $L$ , of the machines, sorted by their potential in nondecreasing order. That is,  $Q_{L[1]} \leq Q_{L[2]} \leq \dots$  (where  $L[k]$  denotes the machine at position  $k$  in  $L$ ). The list  $L$  is updated along the execution of the algorithm. Specifically, when  $M_i$  allocates processing units to some job, its potential decreases, and its position in  $L$  may be updated. Once  $M_i$  has allocated  $u_i w$  processing units, it is removed from  $L$ . Given a pair of machines  $M_{i_1}, M_{i_2}$ , we say that  $M_{i_1}$  is weaker (stronger) than  $M_{i_2}$ , if  $Q_{i_1} \leq Q_{i_2}$  ( $Q_{i_1} \geq Q_{i_2}$ ).

The jobs, sorted in nonincreasing order by their processing ratios, are scheduled one after the other. The job  $J_j$  is scheduled on the first (i.e., weakest) consecutive sequence of  $\rho_j + 1$  or less machines,  $L[k_1], \dots, L[k_2]$ , whose total potential is at least  $t_j$ . All the potential of  $L[k_1], \dots, L[k_2 - 1]$  and some of the potential of  $L[k_2]$  is allocated to  $J_j$ , such that the total number of processing units allocated to  $J_j$  is  $t_j$ . We show that such a sequence of machines always exists. The selection of this sequence is done as follows. We first examine  $L[1]$ , which is the weakest machine. If  $Q_{L[1]} \geq t_j$  we schedule  $J_j$  on  $L[1]$  and update the potential of this machine; else, we test whether  $Q_{L[1]} + Q_{L[2]} \geq t_j$ , and so on until either we find a sequence of machines with sufficient potential ( $\geq t_j$ ), or the total potential of the first  $\rho_j + 1$  machines is less than  $t_j$ . In the latter case we proceed to examine the next *window* of  $\rho_j + 1$  machines,  $L[2], \dots, L[\rho_j + 2]$ , and so on, until our window covers  $\rho_j + 1$  machines,  $L[k], \dots, L[k + \rho_j]$ , such that  $t_j > Q_{L[k-1]} + \dots + Q_{L[k+\rho_j-1]}$  and  $t_j \leq Q_{L[k]} + \dots + Q_{L[k+\rho_j]}$  (see Figure 2). At this stage we can clearly allocate to  $J_j$  all the potential of the machines  $L[k], \dots, L[k + \rho_j - 1]$ , and complete the execution of  $J_j$  by also allocating to it some of the potential of the machine  $L[k + \rho_j]$ .

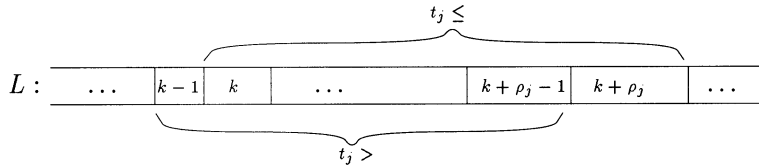


Fig. 2. The window scanning the list  $L$ .

In fact,  $\mathcal{A}_r$  only determines the amount of processing units allocated to each job by each of the machines. The order of the jobs on each machine is arbitrary. Note that  $J_j$  is allotted to at most  $\rho_j + 1$  machines. (Thus, at most  $\rho_j + 1$  machines process it *simultaneously*.)

Let  $J_b$  be the first job such that, when  $J_b$  is scheduled, the list  $L$  contains at most  $\rho_b$  machine indices, or the total processing potential of the  $\rho_b$  weakest machines is at least  $t_b$ . We distinguish between two phases of  $\mathcal{A}_r$ :

1. The jobs  $J_1, \dots, J_{b-1}$  are scheduled.
2. The jobs  $J_b, \dots, J_n$  are scheduled.

We show that in each phase the corresponding set of jobs is scheduled legally. Note that jobs scheduled in the first phase are scheduled using the *moving-window*, each on exactly  $\rho_j + 1$  machines. For this phase, we need to show that we never fail to find a subset of  $\rho_j + 1$  machines that can complete  $J_j$ . Specifically, we show below that for each  $j < b$ , when  $J_j$  is scheduled, the total potential of the strongest  $\rho_j + 1$  machines in  $L$  is at least  $t_j$ . For the second phase of  $\mathcal{A}_r$  we show that for each  $b \leq j \leq n$ ,  $J_j$  is allocated  $t_j$  processing units from at most  $\rho_j + 1$  machines.

For simplicity, assume that whenever we use the moving-window to schedule a job, the list  $L$  is scanned from left to right. That is, the index of the weakest machine,  $L[1]$ , is the leftmost, and the index of the strongest machine is the rightmost in  $L$ . The window moves from left to right until for some  $k$  (which denotes the index in  $L$  of the weakest machine in the window), we get that the  $\rho_j + 1$  machines  $L[k - 1], \dots, L[k + \rho_j]$  can complete the execution of  $J_j$ . When  $J_j$  is scheduled, the machines  $L[k], \dots, L[k + \rho_j - 1]$  are removed from  $L$ , and the machine  $L[k + \rho_j]$  is possibly moved to a new position in the list  $L$ , according to its remaining potential. Note that this new position of  $L[k + \rho_j]$  in  $L$  is left to its original position.

We can view the removed sequence of machines as a *hole* in  $L$ . Each job  $J_j$  creates a hole of  $\rho_j$  machines in  $L$ , and one additional machine (the  $(\rho_j + 1)$ th) is moved left to the hole. By analyzing these holes we conclude that  $\mathcal{A}_r$  never fails to schedule jobs during its first phase.

**LEMMA 3.3.** *Each job,  $J_j$ , scheduled by  $\mathcal{A}_r$  during the first phase, is scheduled on exactly  $\rho_j + 1$  machines and is allocated  $t_j$  processing units.*

**PROOF.** Assume that for some job  $J_g$ , the moving window procedure fails to complete  $J_g$ . That is, the window reaches the rightmost position in  $L$ , but the total potential of the  $\rho_g + 1$  strongest machines covered by the window is less than  $t_g$ .

Let us examine the sequence of holes created in  $L$  by the time  $J_g$  is scheduled. We first show that when we fail to schedule  $J_g$ , its partial schedule creates at the right end of  $L$  a hole, which is the union of holes created by previously scheduled jobs.

**CLAIM 3.4.** *There exists a set  $S \subseteq \{J_1, \dots, J_g\}$ , such that the hole created by  $J_g$  unites the holes created by the jobs in  $S$ , into a single hole positioned at the right end of  $L$ .*

**PROOF.** Consider the hole in  $L$  that contains the  $\rho_g + 1$  machines on which  $J_g$  is scheduled and all the holes that are united when these  $\rho_g + 1$  machines are removed.

Let  $H$  denote this united hole. Since each job  $J_j$ ,  $j < g$ , creates a hole of  $\rho_j$  consecutive machines in  $L$ , then for each such job, either *all* or *none* of these  $\rho_j$  machines is contained in  $H$  (we ignore the  $(\rho_j + 1)$ th machine:  $J_j$  may only partially use this machine, in which case the machine is not removed from  $L$ ).

Let  $S$  be the set of all the jobs,  $J_j$ , that contribute  $\rho_j$  or  $\rho_j + 1$  machines to  $H$ . By definition, the hole  $H$  unites the holes created by the jobs in  $S$  into a single hole. In addition, since  $J_g$  is not completed,  $H$  must include the strongest  $\rho_g + 1$  available machines, and in particular the rightmost one. Therefore,  $H$  is positioned at the right end of  $L$ .  $\square$

We conclude that  $\mathcal{A}_r$  allocates to the jobs in  $S$  at least all the potential of the  $\sum_{J_j \in S} \rho_j$  strongest machines.

CLAIM 3.5. *The total potential of the  $\sum_{J_j \in S} \rho_j$  fastest machines is at least  $\sum_{J_j \in S} t_j$ .*

PROOF. Let  $\rho_S = \sum_{J_j \in S} \rho_j$ . Recall that for any  $1 \leq \ell \leq n$ ,  $\hat{\rho}_\ell = \sum_{j=1}^\ell \rho_j$ . Let  $J_\ell$  be the job such that  $\hat{\rho}_{\ell-1} < \rho_S \leq \hat{\rho}_\ell$ . Note that  $\ell \leq g$  since  $S \subseteq \{J_1, \dots, J_g\}$ . By the definition of  $w$ ,  $w \cdot \sum_{i=1}^{\hat{\rho}_{\ell-1}} u_i \geq \sum_{j=1}^{\ell-1} t_j$  and  $w \cdot \sum_{i=1}^{\hat{\rho}_\ell} u_i \geq \sum_{j=1}^\ell t_j$ . That is, the first  $\hat{\rho}_{\ell-1}$  machines are strong enough to complete the first  $\ell - 1$  jobs, and the first  $\hat{\rho}_\ell$  machines are strong enough to complete the first  $\ell$  jobs. Since the machines are sorted such that  $u_i \geq u_{i+1}$ , we conclude that for any integer  $0 < x \leq \rho_\ell$ , the first  $\hat{\rho}_{\ell-1} + x$  machines can complete the first  $\ell - 1$  jobs and an  $(x/\rho_\ell)$ -fraction from  $J_\ell$ . Formally,  $w \cdot \sum_{i=1}^{\hat{\rho}_{\ell-1}+x} u_i \geq \sum_{j=1}^{\ell-1} t_j + x(t_\ell/\rho_\ell)$ .

In particular, for  $x = \rho_S - \hat{\rho}_{\ell-1}$ , we have that

$$(2) \quad w \sum_{i=1}^{\rho_S} u_i \geq \sum_{j=1}^{\ell-1} t_j + (\rho_S - \hat{\rho}_{\ell-1}) \frac{t_\ell}{\rho_\ell}.$$

For a set,  $Y$ , of jobs  $J_{i_1}, J_{i_2}, \dots$ , consider the vector  $\vec{v}_Y$  consisting of  $\rho_{i_1}$  entries with the value  $t_{i_1}/\rho_{i_1}$ , followed by  $\rho_{i_2}$  entries with the value  $t_{i_2}/\rho_{i_2}$ , and so on. For the set  $\mathcal{J}$  of all the jobs in our instance, consider the vector  $\vec{v}_1$  consisting of the first  $\rho_S$  entries of  $\vec{v}_\mathcal{J}$ . Since the jobs are sorted such that  $t_j/\rho_j \geq t_{j+1}/\rho_{j+1}$ , and since  $S$  is a subset of  $\mathcal{J}$ ,  $\vec{v}_1 \geq \vec{v}_S$ . That is, for any index  $i$ ,  $v_1^i \geq v_S^i$ . Therefore,  $\sum_{i=1}^{\rho_S} v_1^i \geq \sum_{i=1}^{\rho_S} v_S^i$ . However,  $\sum_{i=1}^{\rho_S} v_1^i = \sum_{j=1}^{\ell-1} t_j + (\rho_S - \hat{\rho}_{\ell-1})(t_\ell/\rho_\ell)$ , and  $\sum_{i=1}^{\rho_S} v_S^i = \sum_{J_j \in S} t_j$ . Thus,  $\sum_{j=1}^{\ell-1} t_j + (\rho_S - \hat{\rho}_{\ell-1})(t_\ell/\rho_\ell) \geq \sum_{J_j \in S} t_j$ .

From (2), we get that the  $\rho_S$  strongest machines can complete the jobs in  $S$ .  $\square$

This contradicts our assumption that  $\mathcal{A}_r$  fails to schedule  $J_g \in S$  during the first phase.  $\square$

We turn to show that all the jobs that are scheduled during the second stage of  $\mathcal{A}_r$  are scheduled legally.

LEMMA 3.6. *Each job,  $J_j$ , scheduled by  $\mathcal{A}_r$  during the second phase, is scheduled on at most  $\rho_j + 1$  machines and is allocated  $t_j$  processing units.*

PROOF.  $\mathcal{A}_r$  reaches the second phase if, for the next job to be scheduled,  $J_b$ , the list  $L$  contains at most  $\rho_b$  machine indices, or if the total processing potential of the  $\rho_b$  weakest machines is at least  $t_b$ .

In the analysis of the second phase, we need to show that all the jobs are completed, and that at most  $\rho_j + 1$  machines participate in the schedule of  $J_j$ . For showing that all jobs are completed, note that when the algorithm starts, we have  $w \sum_{i=1}^m u_i \geq \sum_{j=1}^n t_j$ , that is, the total processing potential is at least the total processing requirement of the jobs. This is due to the fact that during the first phase no job is allocated more than  $t_j$  processing units. This guarantees that when we schedule  $J_j$  greedily, we never run out of processing potential.

Clearly, at most  $\rho_b$  machines participate in the greedy schedule of  $J_b$ . The last machine on which  $J_b$  is scheduled may have additional processing potential. This machine,  $M_\ell$ , is now the weakest machine (since it belonged to the set of weakest machines before  $J_b$  was scheduled, and all the weaker machines in this set are now omitted from  $L$ ), i.e.,  $L[1] = \ell$ . We now proceed to schedule the remaining jobs. In order to show that for any  $j > b$ , at most  $\rho_j + 1$  machines share the execution of  $J_j$ , we first prove the following claim.

CLAIM 3.7. *After  $J_b$  is scheduled, for any  $j > b$ , the list  $L$  contains at most  $\rho_j$  machines, or the total potential of the  $\rho_j$  machines  $L[2], \dots, L[\rho_j + 1]$  is at least  $t_j$ .*

PROOF. Assume that  $J_b$  is scheduled on  $x$  machines. Clearly,  $x \leq \rho_b$ ; thus, the potential of the strongest machine among these  $x$  machines is at least  $t_b/x \geq t_b/\rho_b \geq t_j/\rho_j$ , for any  $j > b$ . Following the schedule of  $J_b$ , the list  $L$  is updated, and the weakest  $x - 1$  machines are removed; the remaining potential of the next ( $x$ th) machine becomes  $Q_{L[1]}$ . Since the machines in  $L$  are sorted in nondecreasing order by their potential, the potential of each of the machines  $L[2], \dots, L[\rho_j + 1]$  (assuming  $|L| > \rho_j$ ) is at least  $t_j/\rho_j$ , meaning that the total potential of these  $\rho_j$  machines is at least  $t_j$ .  $\square$

Consider a job  $J_j$ ,  $j > b$ . If, following the schedule of  $J_b$ , the list  $L$  contains at most  $\rho_j$  machines, then, clearly,  $J_j$  will be scheduled on at most  $\rho_j$  machines; otherwise, note that if the  $\rho_j$  machines  $L[2], \dots, L[\rho_j + 1]$  are strong enough to complete  $J_j$ , then any set of  $\rho_j$  machines, not including  $L[1]$ , is strong enough for  $J_j$ .

We show that  $\mathcal{A}_r$  never uses more than  $\rho_j + 1$  machines for processing  $J_j$ . Once  $J_b$  is scheduled, we turn to schedule  $J_{b+1}$ . Consider the subset of the  $\rho_{b+1} + 1$  weakest machines. It consists of  $L[1]$  and additional  $\rho_{b+1}$  machines. From the above discussion, the total potential of the additional  $\rho_{b+1}$  machines is at least  $t_{b+1}$ , therefore (even if  $Q_{L[1]}$  is small), the total potential of the weakest  $\rho_{b+1} + 1$  machines is at least  $t_{b+1}$ , and we can allocate to  $J_{b+1}$  exactly  $t_{b+1}$  processing units, by using at most  $\rho_{b+1} + 1$  machines. Again, the last machine may have remaining potential. The same argument holds for all the remaining jobs. That is, every job  $J_j$  will be allocated exactly  $t_j$  processing units, using at most  $\rho_j + 1$  machines.  $\square$

*Proof of Theorem 3.1.* From Lemmas 3.3 and 3.6 we get that  $\mathcal{A}_r$  assigns to each of the jobs,  $J_j$ ,  $t_j$  processing units, on at most  $\rho_j + 1$  machines. In addition, from Lemma 3.2, the length of the schedule is  $w \leq w_{\text{OPT}}(I)$ .

We now turn to compute the running time of the algorithm. We show that  $\mathcal{A}_r$  can be implemented in  $O(\max(m \lg m, n \lg n))$  steps:  $O(m \lg m) + O(n \lg n)$  steps are required for sorting the lists and calculating  $w$ . Given that the lists are sorted, the total time for scheduling the jobs is  $O(m+n) + O(n \lg m)$ . The first phase of the algorithm, in which we schedule the jobs using the moving-window, can be implemented in  $O(m) + O(n \lg m)$  steps. The idea is to start scanning the list for each job,  $J_j$ , from a fixed point, which depends on  $J_j$ . Recall that in this phase, each job  $J_j$ ,  $j > 1$ , is processed by exactly  $\rho_j + 1$  consecutive machines. This set of machines must contain the strongest machine among those, whose potential is at most  $t_j/(\rho_j + 1)$ . Since  $L$  is sorted, finding this machine can be done (e.g., using skip-lists [16], [15]) in  $O(\lg m)$  steps. We can now find in  $O(\rho_j)$  steps the set of  $\rho_j + 1$  machines that will process  $J_j$ . Finally, after we schedule  $J_j$ , we need to reposition the  $(\rho_j + 1)$ th machine in  $L$ , according to its remaining potential. Since  $L$  is sorted, this can be done in  $O(\lg m)$  steps.

Let  $P_1$  denote the set of jobs scheduled in the first phase. Each of these jobs,  $J_j \in P_1$ , uses up the potential of a set of  $\rho_j$  machines (which are then omitted from  $L$ ). This implies that  $\sum_{J_j \in P_1} \rho_j \leq m$ , and, therefore, the total time required for positioning the window and scheduling the jobs in  $P_1$  is  $O(n \lg m) + O(m)$ .

From Claim 3.7, during the second phase of the algorithm we schedule the jobs greedily. Hence, this phase requires  $O(m+n)$  steps. This completes the proof.  $\square$

**3.1.2. A  $\max_j(1 + 1/\rho_j)$ -Approximation.** The algorithm  $\mathcal{A}_r$  yields the following approximation algorithm,  $\mathcal{A}_1$ , for the SPAC problem. Given an instance,  $I$ :

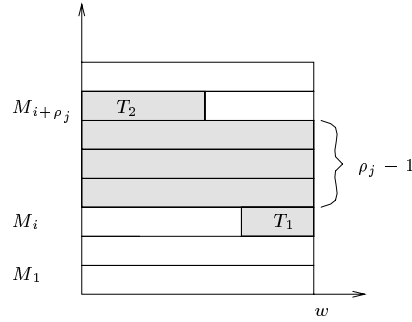
1. Use  $\mathcal{A}_r$  to find a schedule of length  $w$  for  $I^+$ .
2. For each job,  $J_j$ ,  $j = 1, \dots, n$ :
  - If  $J_j$  is scheduled on  $a_j + 1$  machines or if  $\rho_j + 1$  machines process  $J_j$  simultaneously:
    - Let  $M_s$  be the machine which allocated the minimal number of processing units to  $J_j$ .
    - Omit  $J_j$  from the set of jobs scheduled on  $M_s$ .
    - Any other machine,  $M_i$ , which processed  $J_j$  for  $x$  time units, will now process  $J_j$  for  $x(1 + 1/\rho_j)$  time units.

In other words, we transform the infeasible schedule into a feasible one by splitting, for each job,  $J_j$ , the processing of  $J_j$  on the least-contributing machine among the other  $\rho_j$  machines that process  $J_j$ . As shown below, this extends the makespan of the schedule by a fraction which depends on the minimal parallelism parameter of any job.

**THEOREM 3.8.**  $w_{\mathcal{A}_1}(I) \leq \max_j(1 + 1/\rho_j)w_{\text{OPT}}(I)$ .

**PROOF.** We first show that each job,  $J_j$ , is allocated at least  $t_j$  processing units. Clearly, the machine  $M_s$  allocated to  $J_j$  at most  $t_j/(\rho_j + 1)$  processing units. Thus, the other  $\rho_j$  machines allocate to  $J_j$  at least  $\rho_j t_j/(\rho_j + 1)$  processing units. The execution of  $J_j$  on each of these machines is “stretched” by a factor  $1 + 1/\rho_j$ . Hence, the total allocation of processing units to  $J_j$  on these machines is increased to be at least  $(\rho_j t_j/(\rho_j + 1))(1 + 1/\rho_j) = t_j$ .

To bound the resulting makespan, note that, in the worst case, there exists a machine that has to compensate for all the jobs that it executes, meaning that its processing time



**Fig. 3.** The schedule of  $J_j$ .

is stretched from  $w$  to at most  $\max_j w(1 + 1/\rho_j)$ . By Theorem 3.1,  $\mathcal{A}_r$  generates a schedule of length  $w \leq w_{\text{OPT}}(I)$ , thus, the makespan obtained by our algorithm is at most  $\max_j (1 + 1/\rho_j)w_{\text{OPT}}(I)$ .  $\square$

In particular, if the allotment and parallelism limits of each job are at least  $b$ , for some  $b \geq 1$ , that is,  $\rho_j \geq b$  for all  $1 \leq j \leq n$ , the above approximation algorithm yields the makespan  $(1 + 1/b)w_{\text{OPT}}(I)$ .

**3.2. Identical Machines.** We show that for the special case where the machines are identical, the algorithm  $\mathcal{A}_r$  is a  $\max_j (1 + 1/(2\rho_j - 1))$ -approximation to the optimal. Assume without loss of generality that all machines have the rate  $u = 1$ . Thus,  $w = \max\{\sum_j t_j/m, \max_j(t_j/\rho_j)\}$ .

When  $\mathcal{A}_r$  is executed on an instance with identical machines, since  $t_1 \leq w\rho_1$ , we schedule greedily all the jobs. Thus, each job,  $J_j$ , is scheduled on a set of consecutive machines (see Figure 3). Let  $T_1$  and  $T_2$  denote the lengths of the time intervals allocated to  $J_j$  on the “extreme” machines (e.g.,  $M_i$  and  $M_{i+\rho_j}$  in Figure 3). Note that if  $J_j$  is scheduled on  $\rho_j + 1$  machines, then  $t_j = (\rho_j - 1)w + T_1 + T_2$  and  $T_1 + T_2 \leq w$  (otherwise,  $J_j$  is allocated more than  $w\rho_j \geq t_j$  processing units). As in the case of uniform machines, we transform the infeasible schedule into a feasible one, by splitting, for each job,  $J_j$ , the processing of  $J_j$  on the least-contributing machine among the other  $\rho_j$  machines that process  $J_j$ . However, as we show below, the “stretching” factor of each machine can be reduced to  $1 + 1/(2\rho_j - 1)$ . Thus, the resulting algorithm,  $\mathcal{A}_2$ , has a better approximation ratio.

**THEOREM 3.9.**  $w_{\mathcal{A}_2}(I) \leq \max_j (1 + 1/(2\rho_j - 1))w_{\text{OPT}}(I)$ .

**PROOF.** Assume without loss of generality that  $T_1 \leq T_2$ . Since  $T_1 + T_2 \leq w$ , we have that  $T_1 \leq w/2$ . In the “stretched” schedule,  $J_j$  is allocated  $x_j$  processing units.

$$x_j = (w(\rho_j - 1) + T_2) \left( 1 + \frac{1}{2\rho_j - 1} \right) = w(\rho_j - 1) + T_2 + (w(\rho_j - 1) + T_2) \frac{1}{2\rho_j - 1}.$$

Since  $t_j = w(\rho_j - 1) + T_2 + T_1$ , the additional  $(w(\rho_j - 1) + T_2)(1/(2\rho_j - 1))$  processing units need to compensate for the  $T_1$  units that were omitted. Indeed,

$$\begin{aligned} \frac{w(\rho_j - 1) + T_2}{2\rho_j - 1} - T_1 &= \frac{w(\rho_j - 1) + T_2 - (2\rho_j - 1)T_1}{2\rho_j - 1} \\ &\geq \frac{w(\rho_j - 1) + T_1 - (2\rho_j - 1)T_1}{2\rho_j - 1} \\ &= \frac{w(\rho_j - 1) - (2\rho_j - 2)T_1}{2\rho_j - 1} \\ &= \frac{\rho_j - 1}{2\rho_j - 1}(w - 2T_1) \geq 0. \end{aligned}$$

Thus,  $x_j \geq w(\rho_j - 1) + T_2 + T_1 = t_j$ , meaning that  $J_j$  is allocated at least  $t_j$  processing units.  $\square$

**4. A PTAS for Scheduling with Allotment Constraints.** In this section we present a PTAS for the SAC problem on uniform machines. We assume that the maximal allotment parameter of any job is some fixed constant. In Section 2 we have shown that SAC is strongly NP-hard in this case, even for instances with identical machines (Theorem 2.1) or identical jobs (Theorem 2.6).

Our PTAS consists of two stages. In the first stage we guess a partition of each job,  $J_j$ , to at most  $a_j$  segments. In the second stage we consider each job segment as a separate job. The resulting instance has at most  $na_{\max}$  jobs. We run on this instance a PTAS,  $P^*$ , for multiprocessor scheduling on uniform machines (e.g., [12] and [6]). Note that some segments of the same job may be scheduled by  $P^*$  on the same machine: this is equivalent to sharing the execution of  $J_j$  among fewer machines.

An immediate problem which arises when trying to apply this, is that the number of possible partitions is *exponential*. We show that it suffices to examine only a polynomial number of possible partitions in order to approximate the optimal schedule. This subset of partitions can be found and described efficiently.

We first show how to reduce the number of partitions that need to be considered, when the jobs are *identical*. Next, we extend this technique to instances with a *fixed* number of job types. Finally, we show that an *arbitrary* instance can be converted into one in which small jobs cannot split at all, and nonsmall jobs can be replaced by jobs of a fixed number of types. Each of these steps extends the makespan by a factor of  $1 + \varepsilon$ . An additional  $(1 + \varepsilon)$ -extension is caused by  $P^*$ .

**4.1. Identical Jobs.** Assume first that all jobs have the same length,  $t$ , and the same allotment parameter,  $a > 1$ . Thus, the execution of each job can be shared among at most  $a$  machines. Given  $\varepsilon > 0$ , let  $\delta = \varepsilon/a$ .

**LEMMA 4.1.** *Any schedule of  $I$  of length  $C$  can be transformed into one of length at most  $(1 + \varepsilon)C$ , in which all job segments are larger than  $\delta(t/a)$ , and their lengths are multiples of  $\delta^2(t/a)$ .*



PROOF. We say that a job segment is *small* if it has length smaller than  $\delta(t/a)$ . We first describe how the schedule is modified; then we show that the total load on each machine is increased by at most a factor of  $1 + \varepsilon$ . For each job,  $J_j$ :

1. Add all the small segments of  $J_j$  to the longest one and round up the resulting length to the next multiple of  $\delta^2(t/a)$ .
2. Round the length of any other segment to the next multiple of  $\delta^2(t/a)$ .
3. Shorten the longest segment by a multiple of  $\delta^2(t/a)$  such that the total sum of the segment lengths is at least  $t$ .

Clearly, in the resulting schedule all the segments are larger than  $\delta(t/a)$ , their lengths are multiples of  $\delta^2(t/a)$ , and the processing time of each job is at least  $t$ . In addition, since we only group the small segments and add them to the longest one, the allotment constraint is preserved.

We show that for each machine  $M_i$  and job  $J_j$ , if  $M_i$  processed  $J_j$  for  $t_1$  time units, it now processes  $J_j$  for at most  $(1 + \varepsilon)t_1$  time units; thus, the makespan of the schedule is at most  $(1 + \varepsilon)C$ .

For each job, each nonsmall segment, excluding the longest one, has length  $p > \delta(t/a)$  and its new length is at most  $p + \delta^2(t/a) \leq p + \delta p < p(1 + \varepsilon)$ . The longest segment of a job must have length  $p \geq t/a$ . Even if the length of this (long) segment is not reduced in step 3, its new length is now at most  $p + (a - 1)\delta(t/a) + \delta^2(t/a) \leq p + \delta t \leq p + \delta a p = p(1 + \varepsilon)$ .  $\square$

Recall, that our PTAS guesses the partition of the  $n$  jobs, each into at most  $a$  segments, and uses  $P^*$  for the guessed partitions. By Lemma 4.1 we conclude that we pay only  $\varepsilon$  for considering only a subset of the possible partitions. We now show that an optimal partition can be guessed efficiently. That is, the number of possible partitions is polynomial in  $n$ .

Denote by  $S_\delta$  the set of partitions of a number  $t$  into at most  $a$  numbers which are all larger than  $\delta(t/a)$ , and their values are multiples of  $\delta^2(t/a)$ . In the following we compute the size of  $S_\delta$ , denoted by  $h_\delta$ . We use in the computation the next result, given in [4].

LEMMA 4.2. *Let  $f$  be the number of  $g$ -tuples of nonnegative integers such that the sum of tuple coordinates is equal to  $d$ , for some  $d \geq 1$ . Then  $f = \binom{d+g-1}{g-1}$ . If  $d + g \leq \alpha g$ , for some  $\alpha \geq 1$ , then  $f = O(\alpha^{\alpha g})$ .*

Now, we bound  $h_\delta$  in terms of  $a$ .

LEMMA 4.3. *The size of  $S_\delta$  is  $h_\delta = O((2e)^a)$ , where the symbol  $e$  denotes the base of the natural logarithm.*

PROOF. Note that we can describe a partition of a given job to at most  $a$  segments by an  $(a/\delta^2)$ -tuple: each coordinate,  $i$ , gives the number of segments of length  $i \cdot (\delta^2 t)/a$ ,  $1 \leq i \leq a/\delta^2$ ; the sum of the coordinates is at most  $a$ . By Lemma 4.2, taking  $d = a$ ,  $g = a/\delta^2$  and  $\alpha = 1 + \delta^2$ , we get that the number of such tuples is

$$\binom{a + a/\delta^2 - 1}{a/\delta^2 - 1} = O((1 + \delta^2)^{(1+\delta^2)a/\delta^2}) = O((2e)^a).$$

The last equality follows from the standard bound  $(1+x)^{1/x} \leq e$ , for  $0 < x < 1$ , and the assumption that  $(1+\delta^2) \leq 2$ .  $\square$

Each item in  $S_\delta$  describes a partition of a single job. To describe a partition of the  $n$  jobs we use a vector of length  $h_\delta$ , whose  $i$ th entry specifies for how many jobs we adapt the  $i$ th partition vector. The number of possible vectors is less than  $n^{h_\delta}$ .

**4.2. Fixed Number of Job Types.** Assume that there are  $T$  different job types, where  $T \geq 1$  is some constant. All the  $n_k$  jobs of the  $k$ th type,  $1 \leq k \leq T$ , have length  $t_k$  and allotment parameter  $a_k$ . Note that the proof of Lemma 4.1 considers the extension of each segment on each machine *separately*. Thus, choosing  $\delta_k = \varepsilon/a_k$ , we can extend Lemma 4.1 as follows:

**LEMMA 4.4.** *Any schedule of  $I$  of length  $C$  can be transformed into one of length at most  $(1+\varepsilon)C$ , in which all the segments of jobs of the  $k$ th type are larger than  $\delta(t_k/a_k)$  and their lengths are multiples of  $\delta^2(t_k/a_k)$ .*

Let  $h_{\delta_k}$  be the *constant* number of possible partitions of one job of the  $k$ th type. In order to describe a possible partition of the  $n$  jobs we use a vector of length  $h_{\delta_1} + h_{\delta_2} + \dots + h_{\delta_T}$ , whose entries specify how many jobs of each type are partitioned in a certain way. The number of possible vectors is less than  $\prod_{k=1}^T n_k^{h_{\delta_k}} = O(n^{\sum_{i=1}^T h_{\delta_i}})$ .

**4.3. Arbitrary Jobs.** Given an arbitrary instance, our idea is to distinguish between small and large jobs. For the subset of large jobs we pay  $\varepsilon$  in order to convert it into one with a fixed number of job types—for which, as we showed in Section 4.2, we need to examine only a polynomial number of possible partitions. For the small jobs we show that we may pay at most a factor  $\varepsilon$  from reducing all their allotment parameters to one. Any job,  $J_j$ , for which  $a_j = 1$  need not participate in the “guessing partition” process, i.e.,  $J_j$  is given to  $P^*$  as a single segment.

Let  $t_{\max}$  be the maximal length of any job in  $I$ . Let  $a_1$  be the minimal allotment parameter among the jobs with length  $t_{\max}$ . For a given  $\varepsilon$ , we say that a job is *small* if its length is less than  $\varepsilon(t_{\max}/a_1)$ .

**LEMMA 4.5.** *Any schedule of  $I$  of length  $C$  can be transformed into one of length at most  $(1+\varepsilon)C$ , in which all the small jobs are not partitioned at all (that is, have  $a_j = 1$ ).*

**PROOF.** Given a schedule of  $I$ , let  $W_{s_i}$  be the total number of processing units allocated to segments of small jobs on  $M_i$ . We reallocate  $W_{s_1}, W_{s_2}, \dots$  to the small jobs sequentially, starting from the first small job on the first machine. When a small job is completed we move to the next small job; when all the  $W_{s_i}$  processing units are allocated, we complete the active small job and move to the next machine and to the next small job. Clearly, since we allocate at least  $\sum_i W_{s_i}$  processing units, all the small jobs are completed. Also, since we always complete a job on the machine on which its processing starts, we do not split small jobs. Finally, the makespan can increase at most by a factor of  $1+\varepsilon$ : in the

original schedule, some machine must have load at least  $t_{\max}/a_1$ . Thus,  $C \geq t_{\max}/a_1$ . The total load on each machine is extended by at most  $\varepsilon(t_{\max}/a_1)$ . Therefore, the resulting makespan is at most  $C + \varepsilon(t_{\max}/a_1) \leq C + \varepsilon C = (1 + \varepsilon)C$ .  $\square$

From Lemma 4.5, for the purpose of guessing a partition of jobs to segments, we may assume that our instance consists of *large* jobs. Let  $a_{\max}$  be the maximal allotment parameter of any job in  $I$ . We replace the large jobs by  $T = a_{\max}(a_1/\varepsilon^2 - 1/\varepsilon)$  sets of jobs such that the jobs in each set are identical. For  $1 \leq a \leq a_{\max}$ ,  $1/\varepsilon < \ell \leq a_1/\varepsilon^2$ , let  $N_{a,\ell}$  be the number of jobs with  $a_j = a$  and  $t_j \in ((\ell - 1)\varepsilon^2(t_{\max}/a_1), \ell\varepsilon^2(t_{\max}/a_1)]$ .

Our new instance,  $I'$ , consists of  $N_{a,\ell}$  jobs with  $a_j = a$  and  $t_j = \ell\varepsilon^2(t_{\max}/a_1)$ . As in other PTASs that use interval partition (e.g., [12], [6], and [1]), we have

LEMMA 4.6. *Any schedule of  $I$  of length  $C$  can be replaced by a schedule of  $I'$  of length  $(1 + \varepsilon)C$*

PROOF. Each long job from  $I$  with  $a_j = a$  and  $t_j \in ((\ell - 1)\varepsilon^2(t_{\max}/a_1), \ell\varepsilon^2(t_{\max}/a_1)]$ , contributes to  $I'$  a job with the same allotment parameter and of length  $t_j + \Delta$ ,  $\Delta \leq \varepsilon^2(t_{\max}/a_1)$ . This extension of  $\Delta$  can split among the segments of  $J_j$  as follows. For each machine  $M_i$  and job  $J_j$ , if  $M_i$  processes a fraction  $\alpha$  of  $J_j$ ,  $0 \leq \alpha \leq 1$ ,  $M_i$  will now process a fraction  $\alpha$  of the extended job. Summing over all the segments of  $J_j$ , we get that the extended job is fully processed. The job  $J_j$  is long, thus,  $t_j \geq \varepsilon(t_{\max}/a_1)$ . Therefore,  $\alpha(t_j + \Delta) \leq \alpha(t_j + \varepsilon^2(t_{\max}/a_1)) \leq \alpha(t_j + \varepsilon t_j) \leq \alpha t_j(1 + \varepsilon)$ , and the total processing time of each job on each machine is extended by at most a factor of  $1 + \varepsilon$ .  $\square$

We summarize in the next result.

THEOREM 4.7. *The SAC problem with fixed allotment parameters admits a PTAS, whose running time is  $O(n^{(a_{\max}a_1/\varepsilon^2)(2e)^{a_{\max}}})$ .*

PROOF. The PTAS described above consists of four steps:

1. Distinguishing between small and large jobs.
2. Replacing the large jobs by jobs of  $T \leq a_{\max}(a_1/\varepsilon^2)$  sets of identical jobs, as described in Section 4.3.
3. Guessing a partition of the resulting jobs to segments.
4. Running the PTAS  $P^*$  on the resulting sets of segments and small jobs.

By Lemmas 4.4–4.6, and since  $P^*$  is a PTAS for the multiprocessing scheduling problem, we get that each stage may extend the makespan by a factor of  $1 + \varepsilon$ . Without loss of generality we assume that  $\varepsilon < 1$ ; thus, by running these steps with  $\hat{\varepsilon} = \varepsilon/9$  we get a total extension of factor  $1 + \varepsilon$ . As discussed in Section 4.2, in the third step we examine  $O(n^{\sum_{i=1}^T h_{\delta_k}}) = O(n^{a_{\max}(a_1/\varepsilon^2)(2e)^{a_{\max}}})$  partitions. Finally,  $P^*$  can be implemented in  $O(n^{1/\varepsilon^2})$  (see [6]).

Thus, we get that the overall running time is at most  $O(n^{(a_{\max}a_1/\varepsilon^2)(2e)^{a_{\max}} + 1/\varepsilon^2})$  which yields the statement of the theorem.  $\square$

**5. Solving SPAC for Parallel-Dominated Instances.** In this section we consider instances in which  $\forall j, \rho_j < a_j$ . We call such instances *parallel-dominated*. We show that for these instances the SPAC problem is optimally solvable on identical machines. For uniform machines the solvability of our problem depends on the differences  $(a_j - \rho_j)$ . Recall that preemptive scheduling on multiple machines can be viewed as the SPAC problem on parallel-dominated instances, where  $\rho_j = 1$  and  $a_j = m, \forall 1 \leq j \leq n$ . Our results imply that when we bound the number of machines that can share the execution of each job, the preemptive scheduling problem is:

1. Solvable on identical machines with any allotment constraints.
2. Strongly NP-hard on uniform machines and identical jobs, where each job can run on at most two machines.
3. Solvable on uniform machines where the jobs are identical, and each job can run on at least three machines.

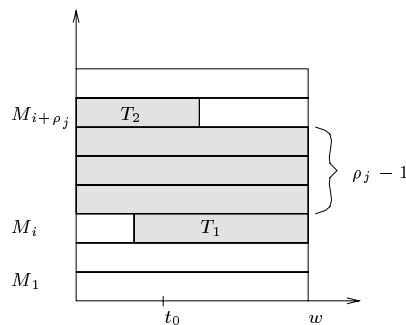
**5.1. An Optimal Algorithm for Identical Machines.** We now show that SPAC is polynomially solvable on identical machines, for any parallel-dominated instance.

**THEOREM 5.1.** *The SPAC problem is solvable in  $O(n + m)$  steps on identical machines, for instances where  $\rho_j < a_j, \forall j$ .*

**PROOF.** Assume without loss of generality that all the machines have the same rate  $u = 1$ . Recall that  $w = \max\{\sum_j t_j/m, \max_j(t_j/\rho_j)\}$  is a lower bound on the length of an optimal schedule.

Consider the simple greedy algorithm,  $\mathcal{A}_u$ , based on McNaughton rule [14].  $\mathcal{A}_u$  proceeds by scheduling the jobs one after the other on the machines. It uses each machine,  $M_i$ , for  $w$  time units, and then moves to  $M_{i+1}$ ; it moves to the next job,  $J_{j+1}$ , once the job  $J_j$  is allocated  $t_j$  processing units. Thus, each job  $J_j$  is scheduled on a consecutive set of machines.

We first show that the parallelism constraints are preserved. Assume by contradiction that there exists a job  $J_j$  such that at time  $t_0 \in [0, w]$ , at least  $\rho_j + 1$  machines process  $J_j$  (see Figure 4).



**Fig. 4.** The schedule of  $J_j$ .

Suppose that the first machine used for processing  $J_j$  is  $M_i$ . At time  $t_0$ , the machines  $M_i, \dots, M_{i+\rho_j}$  process  $J_j$ ; each of the machines  $M_{i+1}, \dots, M_{i+\rho_j-1}$  allocates to  $J_j$ ,  $w$  processing units;  $M_i$  allocates  $T_1$  units and  $M_{i+\rho_j}$  allocates  $T_2$  units. Since both  $M_i$  and  $M_{i+\rho_j}$  process  $J_j$  at time  $t_0$ , we get that  $T_1 + T_2 > w$ . Consequently,  $J_j$  is allocated  $(\rho_j - 1)w + T_1 + T_2 > \rho_j w$  processing units. Since  $w \geq t_j/\rho_j$ , we get that  $J_j$  was allocated more than  $t_j$  processing units, in contradiction to the way  $\mathcal{A}_u$  proceeds.

To see that all the jobs are scheduled within an interval of length  $w$ , note that  $\mathcal{A}_u$  proceeds to the next machine only when the current machine is “saturated.” Thus, it can allocate  $mw$  processing units. By definition,  $w \geq \sum_j t_j/m$ . That is,  $mw \geq \sum_j t_j$ .

Note that in the schedule produced by  $\mathcal{A}_u$ , each job is processed by at most  $\rho_j + 1$  machines. Thus, since  $\forall j, \rho_j < a_j$ , the allotment constraints are preserved.  $\square$

**5.2. Parallel-Dominated Instances and Uniform Machines.** We now show that when the machines may have different speeds, the SPAC problem is strongly NP-hard on parallel-dominated instances, even if all jobs are identical.

**THEOREM 5.2.** *The SPAC problem is strongly NP-hard even if all jobs are identical and  $\forall j, \rho_j < a_j$ .*

**PROOF.** Given an instance of 3-partition, we construct an input,  $I$ , for the makespan problem with  $\forall j, \rho_j = 1, a_j = 2$ , such that  $w_{\text{OPT}}(I) = 1$  if and only if  $A$  has a 3-partition. Note that this is an instance of the preemption problem in which the execution of each job can be shared by at most two machines.

The input  $I$  consists of  $m = 4q$  machines with the following rates: for the first  $3q$  machines,  $u_i = (K + s(x_i))(1 - (K - s(x_i))/(3K - B))^{-1}$ ,  $1 \leq i \leq 3q$ , where  $K > 3qB$  is a large constant. These machines are denoted *slow*. The other  $q$  machines are *fast* with  $u_i = 3K - B$ ,  $3q < i \leq 4q$ . There are  $n = 3q$  identical jobs with  $t_j = 2K$ ,  $\rho_j = 1, a_j = 2, \forall 1 \leq j \leq 3q$ .

**CLAIM 5.3.** *For any slow machine,  $M_i, 1 \leq i \leq 3q, \frac{1}{2}u_i < K$ .*

**PROOF.** By definition of the 3-partition problem,  $\forall x \in A, B/4 < s(x) < B/2$ . Thus,

$$\frac{1}{2}u_i < \frac{1}{2} \left( K + \frac{B}{2} \right) \left( 1 - \frac{K - B/4}{3K - B} \right)^{-1} < \frac{1}{2} \left( K + \frac{B}{2} \right) \frac{5}{3} = \frac{5}{6}K + \frac{5}{12}B < K.$$

The last inequality follows from fact that  $K$  was selected such that  $3B < K$ , therefore,  $\frac{5}{12}B < \frac{1}{2}B < \frac{1}{6}K$ .  $\square$

Assume that  $A$  has a 3-partition to the sets  $S_1, S_2, \dots, S_q, \forall 1 \leq k \leq q$ , let  $S_k = \{x_{k_1}, x_{k_2}, x_{k_3}\}$ . The following is a schedule whose makespan equals 1:  $\forall 1 \leq k \leq q$ , the four machines  $M_{k_1}, M_{k_2}, M_{k_3}, M_{3q+k}$  process the three jobs  $J_{k_1}, J_{k_2}, J_{k_3}$ . Specifically, as illustrated in Figure 5,  $M_{k_i}$  executes  $J_{k_i}, 1 \leq i \leq 3$ , for  $1 - (K - s(x_{k_i}))/ (3K - B)$  time units and thus allocates to it  $K + s(x_{k_i})$  processing units. The fast machine  $M_{3q+k}$  executes  $J_{k_i}, 1 \leq i \leq 3$ , for  $(K - s(x_{k_i}))/ (3K - B)$  time units and thus allocates to it  $K - s(x_{k_i})$  processing units. Since  $s(x_{k_1}) + s(x_{k_2}) + s(x_{k_3}) = B$ , and  $u_{3q+k} = 3K - B$ ,

$M_{k_1}$	idle	$J_{k_1}$	
$M_{k_2}$	$J_{k_2}$	idle	$J_{k_2}$
$M_{k_3}$	$J_{k_3}$		idle
	⋮		
$M_{3q+k}$	$J_{k_1}$	$J_{k_2}$	$J_{k_3}$

Fig. 5. The schedule of  $J_{k_1}, J_{k_2}, J_{k_3}$ .

the completion time equals 1. Note that the allotment constraints are preserved: the execution of  $J_j$  is shared by the two machines  $M_j$  and  $M_{3q+k}$  such that  $x_j \in S_k$ . Also, the parallelism constraints are satisfied:  $M_{k_i}$  is idle while  $J_{k_i}$  is executed on  $M_{3q+k}$ . Finally,  $J_{k_i}$  is allocated exactly  $t_{k_i}$  processing units, since  $(3K - B)(K - s(x_{k_i}))/ (3K - B) + u_{k_i}(1 - (K - s(x_{k_i}))/ (3K - B)) = 2K$ .

Now, suppose that we have a schedule of  $I$  whose makespan equals 1. We show that  $A$  has a 3-partition. First, note that any job,  $J_j$ , has to be scheduled on at least one fast machine. This is due to the fact that  $\rho_j = 1$  and, by Claim 5.3,  $u_i < t_j$  for any slow machine; thus, any combination of slow machines can provide to  $J_j$  less than  $t_j$  processing units. Since  $J_j$  is scheduled on at least one fast machine and  $a_j = 2$ ,  $J_j$  is scheduled on at most one slow machine. Let  $t_{f_j}$  be the total time allocated to  $J_j$  on fast machines, and let  $t_{i_j}$  be the time allocated to  $J_j$  on (at most one) slow machine. Denote by  $M_{i_j}$  the slow machine processing  $J_j$  (if  $J_j$  is processed only by fast machines, then  $t_{i_j} = 0$  and  $M_{i_j}$  is undefined). Let  $x_{i_j}$  be the item with the index  $i_j$  in the input for the 3-partition problem.

LEMMA 5.4. For any job,  $J_j$ , the processing of  $J_j$  is shared by one fast machine and one slow machine,  $M_{i_j}$  such that  $t_{f_j} = (K - s(x_{i_j}))/ (3K - B)$  and  $t_{i_j} = 1 - (K - s(x_{i_j}))/ (3K - B)$ .

PROOF. By definitions of  $t_{f_j}, t_{i_j}$ , and  $M_{i_j}$ ,

$$(3) \quad \begin{cases} t_{f_j} + t_{i_j} \leq 1, \\ (3K - B)t_{f_j} + (K + s(x_{i_j})) \left(1 - \frac{K - s(x_{i_j})}{3K - B}\right)^{-1} t_{i_j} = 2K. \end{cases}$$

CLAIM 5.5. For any job,  $J_j, t_{f_j} \geq (K - s(x_{i_j}))/ (3K - B)$ .

PROOF. Since  $\rho_j = 1, \forall j$ , for any  $\varepsilon > 0$ , if  $t_{f_j} = (K - s(x_{i_j}))/ (3K - B) - \varepsilon$ , then from (3) we get that  $t_{i_j} \leq 1 - (K - s(x_{i_j}))/ (3K - B) + \varepsilon$ , and the overall number of processing units allocated to  $J_j$  is less than  $2K$ .  $\square$

Recall that there are  $3q$  jobs and  $3q$  slow machines. We show now that each job is processed by exactly one slow machine, and that each slow machine processes exactly one job. This is proved by the following claim:

CLAIM 5.6. For all  $j$ ,  $t_{ij} > \frac{1}{2}$ .

PROOF. Assume by contradiction that  $t_{ij} \leq \frac{1}{2}$  for  $y > 0$  jobs. Denote by  $S_1$  the set of  $3q - y$  remaining jobs, i.e.,  $S_1 = \{j \mid t_{ij} > \frac{1}{2}\}$ . Then the jobs in  $S_1$  are scheduled on exactly  $3q - y$  slow machines, each running a single job in this set. Since  $\forall x \in A$ ,  $s(x) > B/4$ , we have  $\sum_{j \in S_1} s(x_{ij}) < Bq - y(B/4)$ . From (3), for any job,  $(3K - B)t_{fj} + u_{ij}t_{ij} = 2K$ ; thus, for each of the  $y$  jobs not in  $S_1$ ,  $t_{fj} \geq (2K - \frac{1}{2}u_{ij})/(3K - B)$ . Also, by Claim 5.5 for each of the  $3q - y$  jobs in  $S_1$ ,  $t_{fj} \geq (K - s(x_{ij}))/ (3K - B)$ .

Summing up the time intervals allocated to all jobs on the fast machines, we get that

$$\begin{aligned} \sum_{j=1}^{3q} t_{fj} &\geq \sum_{j \in S_1} \frac{K - s(x_{ij})}{3K - B} + \sum_{j \notin S_1} \frac{2K - \frac{1}{2}u_{ij}}{3K - B} \\ &\geq \frac{1}{3K - B} \left( (3q - y)K - Bq + y\frac{B}{4} + 2Ky - \sum_{j \notin S_1} \frac{1}{2}u_{ij} \right) \\ &= \frac{1}{3K - B} \left( 3Kq - Bq - Ky + y\frac{B}{4} + 2Ky - \sum_{j \notin S_1} \frac{1}{2}u_{ij} \right) \\ &= q + \frac{1}{3K - B} \left( y \left( K + \frac{B}{4} \right) - \sum_{j \notin S_1} \frac{1}{2}u_{ij} \right) > q. \end{aligned}$$

The last inequality follows from Claim 5.3 and from the assumption that there are  $y > 0$  jobs for which  $t_{ij} \leq \frac{1}{2}$ . However, since the makespan is 1, the total execution time on the fast  $q$  machines cannot exceed  $q$ . Thus,  $y$  equals zero and  $\forall j$ ,  $t_{ij} > \frac{1}{2}$ .  $\square$

It follows that each slow machine processes a different job. Hence,  $\sum_{j=1}^{3q} s(x_{ij}) = \sum_{i=1}^{3q} s(x_i) = qB$ . Assume by contradiction that for some  $j$ ,  $t_{fj} > (K - s(x_{ij}))/ (3K - B)$ . Summing up the time intervals allocated on the fast machines, we get that

$$\sum_{j=1}^{3q} t_{fj} > \frac{3qK - \sum_j s(x_{ij})}{3K - B} = \frac{3qK - qB}{3K - B} = q,$$

which is, again, a contradiction to the length of the schedule.

Assigning  $t_{fj} = (K - s(x_{ij}))/ (3K - B)$  in (3), we get that  $\forall j$ ,  $t_{ij} = 1 - (K - s(x_{ij}))/ (3K - B)$ .  $\square$

Thus, in any schedule whose makespan equals 1,  $J_j$  is allocated exactly  $K + s(x_{ij})$  processing units on some slow machine,  $M_i$ , and exactly  $K - s(x_{ij})$  on some fast machine. Given that each slow machine processes exactly one job, assume without loss of generality that  $\forall 1 \leq i \leq 3q$ ,  $M_i$  processes only  $J_i$ . Also, note that each fast machine must have load  $3K - B$  since the total load on the  $q$  identical fast machines is  $\sum_{i=1}^{3q} K - s(x_i) = (3K - B)q$ .

Denote by *fast-segment* the part of a job that is processed on a fast machine. Consider a fast machine  $M_i, i > 3q$ . Since  $K$  was selected such that  $K \gg s(x), \forall x \in A, M_i$  has to process exactly three fast-segments (Otherwise, there exists a fast machine,  $M_i$ , which processes at least four fast-segments, but any four fast-segments require more than  $u_i = 3K - B$  processing units). For each  $3q < k \leq 4q$ , let  $J_{k_1}, J_{k_2}$ , and  $J_{k_3}$  be the three jobs whose fast-segments are scheduled on  $M_{3q+k}$ . It follows that  $3K - B = 3K - (s(x_{k_1}) + s(x_{k_2}) + s(x_{k_3}))$ , meaning that  $x_{k_1}, x_{k_2}$ , and  $x_{k_3}$  form a triple for the partition. Since all the jobs are scheduled, the schedule of the fast-segments on the fast machines induces a valid 3-partition of  $A$ .  $\square$

In the case where the jobs are identical, and  $\forall j, \rho_j < a_j - 1$ , an optimal algorithm exists.

**THEOREM 5.7.** *The SPAC problem has an  $O(m \lg m)$  optimal algorithm when the jobs are identical and  $\forall j, \rho_j < a_j - 1$ .*

**PROOF.** Assume that for all jobs  $t_j = t, \rho_j = \rho, a_j = a$ , where  $a \geq \rho + 2$ . Let  $w = nt / \sum_{i=1}^m u_i$ . By Lemma 3.2,  $w$  is a lower bound for  $w_{\text{OPT}}(I)$ . We give an algorithm,  $\mathcal{A}_d$ , which outputs a schedule of  $I$  of length  $w$ ; each job is scheduled on at most  $\rho + 2$  machines.

For each machine,  $M_i$ , let  $Q_{M_i}$  denote the *potential* of  $M_i$ , that is, the number of processing units that  $M_i$  can still allocate. Initially, for each of the  $m$  machines  $Q_{M_i} = u_i w$ . Generally,  $Q_{M_i} = u_i w'$ , where  $w'$  is the total length of intervals in  $[0, w]$  in which  $M_i$  is idle. Given a pair of machines  $M_1, M_2$ , we say that  $M_1$  is weaker (stronger) than  $M_2$ , if  $Q_{M_1} \leq Q_{M_2}$  ( $Q_{M_1} \geq Q_{M_2}$ ).

**DEFINITION 5.1.** A *merged machine*,  $M'$ , is a pair of machines  $M_{i_1}, M_{i_2}$  such that  $i_1 < i_2$ , and the machine  $M_{i_1}$  is idle exactly from 0 to  $w_0$ ,  $M_{i_2}$  is idle exactly from  $w_0$  to  $w$ , for some  $w_0 \in [0, w]$ . The potential of a merged machine is  $Q_{M'} = u_{i_1} w_0 + u_{i_2} (w - w_0)$ .

Similar to the algorithm  $\mathcal{A}_r$ , described in Section 3.1, the algorithm  $\mathcal{A}_d$  maintains a list,  $L$ , of the machines, sorted by their potential in nondecreasing order. That is,  $Q_{L[1]} \leq Q_{L[2]} \leq \dots (L[k]$  denotes the machine at position  $k$  in  $L$ ). The list  $L$  is updated along the algorithm, according to the current available machines.

The jobs are scheduled one after the other. We first describe the schedule of  $J_1$ , and then the schedule of  $J_j, j > 1$ . The job  $J_1$  is scheduled on a consecutive set of machines, selected as follows. First, we examine  $L[1]$ , which is the weakest machine. If  $Q_{L[1]} \geq t$  we schedule  $J_1$  on  $L[1]$  and update the potential of this machine; else, we check whether  $Q_{L[1]} + Q_{L[2]} \geq t$ , and so on until either we find a sequence of at most  $\rho$  machines with sufficient potential ( $\geq t$ ), or the total potential of the first  $\rho$  machines is less than  $t$ . In the former case, we schedule  $J_1$  greedily on the weakest machines until it is completed. In the latter case, we proceed to examine the next *window* of  $\rho$  machines,  $L[2], \dots, L[\rho + 1]$ , and so on, until our window covers  $\rho$  machines,  $L[k], \dots, L[k + \rho - 1]$ , such that  $Q_{L[k-1]} + \dots + Q_{L[k+\rho-2]} < t$  and  $Q_{L[k]} + \dots + Q_{L[k+\rho-1]} \geq t$  (see Figure 6). At this stage, we allocate to  $J_1$  all the potential of the



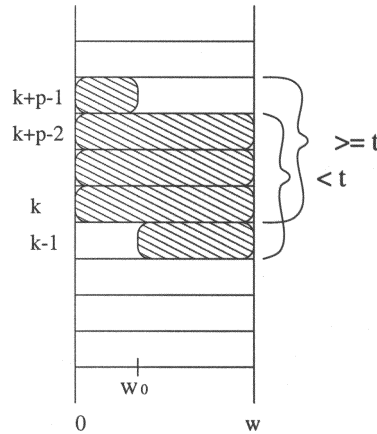


Fig. 6. Scheduling the first job.

machines  $L[k], \dots, L[k + \rho - 2]$ , and complete the execution of  $J_1$  by also allocating to it some of the potential of the machines  $L[k - 1]$  and  $L[k + \rho - 1]$  in the following way. Note that we have  $w(\sum_{k-1 \leq i \leq k+\rho-2} u_{L[i]}) < t$  and  $w(\sum_{k \leq i \leq k+\rho-1} u_{L[i]}) \geq t$ ; thus,  $wu_{L[k-1]} < t/\rho$  and  $wu_{L[k+\rho-1]} \geq t/\rho$ , and there exists some  $w_0 \in [0, w]$  such that the execution of  $J_1$  can be completed if it is scheduled on  $L[k - 1]$  in the interval  $[w_0, w]$  and on  $L[k + \rho - 1]$  in the interval  $[0, w_0]$  (see Figure 6). Therefore,  $J_1$  is scheduled on  $\rho + 1$  machines, and, in each moment, at most  $\rho$  machines process it simultaneously.

The machines  $Q_{L[k]}, \dots, Q_{L[k+\rho-2]}$  are removed from  $L$ . The two machines  $L[k - 1]$  and  $L[k + \rho - 1]$  are partially used and form together one *merged machine*,  $M'$ , which is idle in  $[0, w]$  and whose potential is  $Q_{M'} = u_{L[k-1]}w_0 + u_{L[k+\rho-1]}(w - w_0)$ . This merged machine replaces  $L[k - 1]$  and  $L[k + \rho - 1]$  in  $L$ .

For any job  $J_j, j > 1$ : if the total potential of the weakest  $\rho$  machines is at least  $t$ , then  $J_j$  and all the remaining jobs are scheduled greedily starting from the weakest machine; otherwise, as in the schedule of  $J_1$ , we scan the list  $L$  using a moving-window that covers  $\rho$  machines, until we find the weakest consecutive set of  $\rho$  machines that can complete the execution of  $J_j$  (the merged machine is considered as a single machine in  $L$ ). We show that at any stage of the algorithm, the list  $L$  contains at most one merged machine.

Let  $J_b$  be the first job such that, when  $J_b$  is scheduled, the total potential of the weakest  $\rho$  machines is at least  $t$  (that is,  $J_b, \dots, J_n$  are scheduled greedily). We distinguish between two phases of  $\mathcal{A}_d$ :

1. The jobs  $J_1, \dots, J_{b-1}$  are scheduled.
2. The jobs  $J_b, \dots, J_n$  are scheduled.

CLAIM 5.8. For all  $1 < j \leq b$ , before  $J_j$  is scheduled, the list  $L$  contains at most one merged machine composed of idle-segments of two machines,  $M_{i_1}$  and  $M_{i_2}$ , such that  $wu_{i_1} < t/\rho$  and  $wu_{i_2} \geq t/\rho$ .

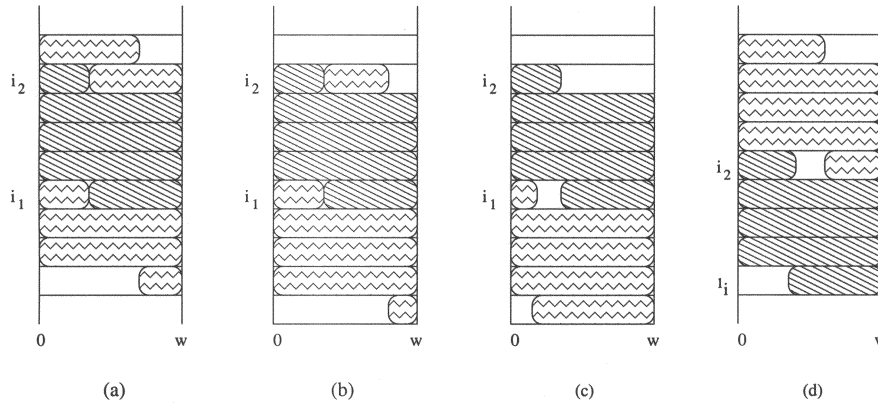


Fig. 7. Scheduling a job  $J_j$ ,  $j > 1$ .

PROOF. The proof is by induction on  $j$ . For  $j = 2$ , we showed that the claim holds after  $J_1$  is scheduled. For the induction step, assume that the claim holds for all  $j \leq k$ . Let  $M_{i_1}$  and  $M_{i_2}$  be the two machines composing the merged machine before  $J_k$  is scheduled. We show that the schedule of  $J_k$  must include some intervals on both  $M_{i_1}$  and  $M_{i_2}$ . As illustrated in Figure 7(a),(b), the only possible way to schedule  $J_k$  is to select a set of machines that encircle the “hole” created by the machines that were previously used, and omitted from  $L$  (the hole is shaded with lines). This follows from the fact that  $J_k$  is scheduled on the first possible consecutive set of machines in  $L$ . Assume that  $J_k$  is scheduled only on machines that are weaker than  $M_{i_1}$  (Figure 7(c)). By the induction hypothesis  $wu_{i_1} < t/\rho$  and thus the execution of  $J_k$  is done solely on machines whose rates are less than  $t/\rho$ . Clearly, no combination of such  $\rho$  machines can complete  $J_k$  in  $w$  time units. Similarly, the schedule of  $J_k$  cannot be done solely on machines which are stronger than  $M_{i_2}$  (Figure 7(d)), since  $wu_{i_2} \geq t/\rho$ . We note that if  $wu_{i_2} = t/\rho$  and the  $\rho$  machines following  $M_{i_2}$  in  $L$  have the rate  $t/w\rho$ , then we may not schedule  $J_j$  on  $M_{i_1}$ , but in this case the schedule of  $J_k$  will be concatenated to that of  $J_{k-1}$  and the claim holds.

Given that the window contains both  $M_{i_1}$  and  $M_{i_2}$ , we get that the new merged machine must replace the old one. The new merged machine is composed of two machines, such that the slower one has rate at most  $u_{i_1}$  and the faster one has rate at least  $u_{i_2}$ , therefore the claim holds also before the schedule of  $J_{k+1}$ .  $\square$

We turn now to consider the number of machine allotments of a job  $J_j$ ,  $j > 1$ . If  $J_j$  is scheduled greedily, then at most  $\rho$  machines in  $L$  share its execution. By Claim 5.8, at most one of these machines may be a merged machine; hence, we get that  $J_j$  is scheduled on at most  $\rho + 1$  machines (no merged machines are composed during the second phase). Assume that  $J_j$  is scheduled in the first phase, using the moving-window. Let  $L[k], \dots, L[k + \rho - 1]$  be the set of  $\rho$  machines such that  $Q_{L[k-1]} + \dots + Q_{L[k+\rho-2]} < t$  and  $Q_{L[k]} + \dots + Q_{L[k+\rho-1]} \geq t$ . As in the schedule of  $J_1$ , we now allocate to  $J_j$  all the potential of the machines  $L[k], \dots, L[k + \rho - 2]$ , and complete its execution by also allocating to it some of the potential of the machines  $L[k - 1]$  and  $L[k + \rho - 1]$ . This allocation is done such that the idle intervals of  $L[k - 1]$  and  $L[k + \rho - 1]$  form a merged

machine. Overall, at most  $\rho + 1$  machines from  $L$  participate in the execution of  $J_j$ . By Claim 5.8 at most one of these machines may be a merged one, thus  $J_j$  is scheduled on at most  $\rho + 2$  machines.

Also, the parallelism constraint of  $J_j$  is preserved: if  $J_j$  is scheduled in the first phase, then it runs on  $\rho - 1$  machines  $L[k], \dots, L[k + \rho - 2]$  in the interval  $[0, w]$ ; also,  $J_j$  is scheduled on  $L[k - 1]$  and  $L[k + \rho - 1]$  in non-overlapping time intervals. Finally, if one of the machines is “merged,” then  $J_j$  is scheduled on the corresponding two machines in two distinct time intervals. It follows that at any time,  $J_j$  is processed in parallel by at most  $\rho$  machines. The argument is similar for jobs  $J_j$  that were scheduled greedily.

Finally, we show that we never fail to schedule a job, that is, the set of strongest  $\rho$  machines in  $L$  can always complete the execution of a job. Recall that  $w = nt / \sum_{i=1}^m u_i$ . Therefore, for each  $1 \leq j \leq n$ ,  $w \geq jt / \sum_{i=1}^{j\rho} u_i$  (assuming that  $u_1 \geq u_2 \geq \dots \geq u_m$ ). This implies that for any  $j$ , the set of first  $j$  jobs can be completed by the  $j\rho$  fastest machines. In terms of  $\mathcal{A}_d$ , the window never has to move beyond the  $\rho$  fastest available machines.

The algorithm  $\mathcal{A}_d$  can be implemented in time  $O(\max(m \lg m, n))$ .  $O(m \lg m)$  steps are needed for sorting the machines. Next, the total time for scheduling the jobs is  $O(m + n)$ . Recall that the schedule of each job  $J_j$  is done on machines that encircle in  $L$  the hole created by the machines that process  $J_{j-1}$ . Hence, we do not need to scan the list  $L$  from the beginning for each job. Also, when the window covers the  $\rho_j + 1$  machines that will share the execution of  $J_j$ , the calculation of the new merged machine and the updates in  $L$  take  $O(1)$ .  $\square$

**CONCLUDING REMARK.** Determining the minimal difference  $(a_j - \rho_j)$  required by any efficient algorithm for *general* instances remains an open question. In particular, can we optimally solve the classic preemptive scheduling problem when  $\forall j, a_j = 3$  and for larger *constant* values of  $a_j$ ?

**Acknowledgment.** We thank an anonymous referee, whose careful reading of the manuscript has led to significant improvements in the accuracy and clarity of the presentation.

## References

- [1] N. Alon, Y. Azar, G. J. Woeginger and T. Yadid. Approximation schemes for scheduling. In *Proc. of the 10th ACM–SIAM Symposium on Discrete Algorithms (SODA)*, pp. 493–500, 1997.
- [2] J. Blazewick, M. Drabowski, and J. Weglarz. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Transactions on Computers*, 35(C):389–393, 1986.
- [3] P. Brucker and S. Knust. Complexity results of scheduling problems. <http://www.mathematik.uni-osnabrueck.de/research/OR/class/>.
- [4] C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. In *Proc. of the 11th ACM–SIAM Symposium on Discrete Algorithms (SODA)*, pp. 213–222, 2000.
- [5] W.J. Davis, D.L. Setterdahl, J.G. Macro, V. Izokaitis and B. Bauman. Recent Advances in the Modeling, Scheduling and Control of Flexible Automation. In *Proc. of the 1993 Winter Simulation Conference*, San Diego, CA, pp. 143–155, 1993.

- [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*, LNCS 1643, pp. 151–162. Springer-Verlag, Berlin, 1999.
- [7] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.
- [8] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation algorithms for data placement on parallel disks. In *Proc. of the 11th ACM–SIAM Symposium on Discrete Algorithms (SODA)*, pp. 223–232, 2000.
- [9] T. Gonzalez and S. Sahni. Preemptive scheduling of uniform processor systems. *Journal of the ACM*, 25:92–101, 1978.
- [10] L.A. Hall. Approximation algorithms for scheduling. In *Approximation Algorithms for NP-Hard Problems* (D.S. Hochbaum, ed.), chapter 1, pp. 1–45. PWS, Boston, MA, 1995.
- [11] 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.
- [12] D.S. Hochbaum and D.B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. *SIAM Journal of Computing*, 17(3):539–551, 1988.
- [13] E.G. Horvath, S. Lam, and R. Sethi. A level algorithm for preemptive scheduling. *Journal of the ACM*, 24:32–43, 1977.
- [14] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6:1–12, 1959.
- [15] J.I. Munro, T. Papadakis, and R. Sedgewick. Deterministic skip lists. In *Proc. of the Third ACM–SIAM Symposium on Discrete Algorithms*, pp. 367–375, 1992.
- [16] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.
- [17] H. Shachnai and T. Tamir. Polynomial time approximation schemes for class-constrained packing problems. In *Proc. of the Third International Workshop on Approximation Algorithms*, pp. 238–249, 2000.
- [18] H. Shachnai and T. Tamir. On two class-constrained versions of the multiple knapsack problem. *Algorithmica*, 29:442–467, 2001.
- [19] H. Shachnai and T. Tamir. Preemptive scheduling of parallelizable jobs. Manuscript, 2001.
- [20] H. Shachnai and J. Turek. Multiresource malleable task scheduling. *Information Processing Letters*, 70:47–52, 1999.
- [21] A. Silberschatz and P. Galvin. *Operating System Concepts*, 5th edn. Addison-Wesley, Reading, MA, 1998.