Scheduling with Unexpected Machine Breakdowns

Susanne Albers* Günter Schmidt†

Abstract

We investigate an online version of a basic scheduling problem where a set of jobs has to be scheduled on a number of identical machines so as to minimize the makespan. The job processing times are known in advance and preemption of jobs is allowed. Machines are *non-continuously* available, i.e., they can break down and recover at arbitrary time instances *not known in advance*. New machines may be added as well. Thus machine availabilities change online.

We first show that no online algorithm can construct optimal schedules. We also show that no online algorithm can achieve a bounded competitive ratio if there may be time intervals where no machine is available. Then we present an online algorithm that constructs schedules with an optimal makespan of $C_{\max}^{OPT}$ if a *lookahead* of one is given, i.e., the algorithm always knows the next point in time when the set of available machines changes. Finally we give an online algorithm without lookahead that constructs schedules with a nearly optimal makespan of $C_{\max}^{OPT} + \epsilon$, for any $\epsilon > 0$, if at any time at least one machine is available. Our results demonstrate that not knowing machine availabilities in advance is of little harm.

1 Introduction

In scheduling theory the basic model assumes that a fixed set of machines is continuously available for processing throughout the planning horizon. This assumption might be justified in some cases but it does not apply if certain maintenance requirements, breakdowns or other constraints that cause the machines not to be available for processing have to be considered.

Machine availability constraints appear very often. Clearly, machines may be faulty and break down. Moreover, availability constraints arise on the operational level of production scheduling. Here some jobs are fixed in terms of starting and finishing times and resource assignment. When new jobs become available for processing, there are already jobs assigned to time intervals and corresponding machines while the new ones have to be processed using the remaining free processing intervals. A similar problem occurs in operating systems for single- and multi-processors when subprograms with higher priority have to be scheduled before subprograms with lower priority.

*Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. E-mail: albers@mpi-sb.mpg.de. Part of this work was done while visiting the Freie Universität Berlin.

†Information and Technology Management, University of Saarland, 66041 Saarbrücken, Germany. Email: gs@itm.uni-sb.de. Part of this work was done while visiting the ICSI Berkeley. This research was partially supported by INTAS (Project INTAS-96-0812).

Thus, limited machine availability is common in practice. Knowledge about machine availabilities might be complete or incomplete. In an *online setting* machine availabilities are not known in advance. Machine breakdowns are a typical example of events that arise online. Sometimes a scheduler has partial knowledge of the availabilities, i.e, he has some *lookahead*. He might know of the next time interval where a machine requires maintenance or he might know when a broken machine will be available again. In an *offline setting* all machine availabilities are known prior to schedule generation.

In this paper we study a very basic scheduling problem with respect to limited machine availability: A set of jobs has to be scheduled on a set of identical machines so as to minimize the makespan. More specifically, let $\mathcal{J} = \{J_i | i = 1, \ldots, n\}$ be a set of independent jobs to be scheduled. Job J_i has a processing time of p_i time units known in advance, $1 \leq i \leq n$. The jobs have to be scheduled on a set of machines that operate with the same speed. At any time preemption of jobs is allowed at no penalty. Also, the minimum time slice for preemption may be arbitrarily small. The current state of a preempted job is saved for the machine system. If a job is preempted, then it may be resumed later on a any machine. Each machine may work only on one job at a time, and each job may be processed by only one machine at a time. We wish to minimize the *makespan*, i.e., the completion time of the last job that finishes. Machines may have different time intervals of availability. We emphasize here that we are interested in the online version of the problem where the machine availabilities are not known in advance. We also call an interval where a machine is not available a *machine break down*. Machines may break down or recover at arbitrary time instances. New machines may be added as well. If a machine breaks down, then the job currently being processed is simply preempted. We also consider the online problem with *lookahead one*, i.e., a scheduler always knows the next point in time where the set of available machines changes. However, he does not have to know which machines break down or become available. In the previous literature [4, 6], this setting is also referred to as *nearly online*.

Given a scheduling algorithm A and a problem instance, let $C^A_{\max}$ denote the makespan of the schedule produced by A. In particular, $C^{OPT}_{\max}$ denotes the makespan of an optimal offline algorithm that knows the machine availabilities in advance. Following [9] we call an online scheduling algorithm A *c-competitive* if, for all problem instances, $C^A_{\max} \leq c \cdot C^{OPT}_{\max}$.

Related work: Schmidt [7] was the first who studied scheduling problems with limited machine availability. He concentrated on the offline version of the above problem when all the machine breakdown times are known in advance. Note that if the down times are identical for all the machines, then an optimal schedule can be constructed using McNaughton's algorithm [3]. The algorithm runs in $O(n)$ time and uses no more than $S - 1$ preemptions, where S is the total number of intervals where machines are available. Schmidt [7] studied the problem with arbitrary machine availabilities and gave an algorithm that always constructs an optimal schedule. His algorithm has a running time of $O(n + m \log m)$ and uses at most $S - 1$ preemptions if the intervals of availability are rearranged such that they form a staircase pattern. Again, S is the total number of intervals where machines are available. In [8] the problem is generalized taking into account different job release times or deadlines.

There are results for nearly online problems, i.e., the next point in time when a machine breaks down or recovers is known. In [4], Sanlaville presents an algorithm for the problem variant that jobs have release and due dates and the goal is to minimize maximum lateness. At any point in time, the algorithm also has to know the next release date. The algorithm constructs optimal schedules for zigzag machine availability patterns (only m or $m-1$ machines are available at any point in time) but not for arbitrary patterns. The running time of the algorithm is $O(n^2 p_{\max} + T^{up})$, where $p_{\max}$ is the longest processing time of the jobs and T^{up} is the total time needed to *up*date the set of available machines. Sanlaville [4] also reports that his algorithm constructs optimal schedules for arbitrary availability patterns if there are no release dates and the objective to minimize the makespan. However, neither the paper nor a private communication [5] contains any additional information regarding the optimality proof.

As for the online setting, scheduling with unexpected machine breakdowns was studied by Kalyanasundaram and Pruhs [1, 2]. In [1] online algorithms with optimal competitive ratios are given for various numbers of faulty machines. The authors assume that if a machine breaks down, the job currently being processed has to be restarted later from the beginning. Also two specific types of breakdowns are considered. In a *permanent* breakdown a machine does not recover again; in a *transient* breakdown the machine is available again right after the breakdown. This is different from the problem setting we consider. In [2] Kalyanasundaram examine to which extent redundancy can help in online scheduling with faulty machines.

Our contribution: In this paper we study the scheduling problem defined above. As mentioned before we are mainly interested in the online version of the problem. In Section 2 we prove that no online algorithm can construct optimal schedules if machines can break down and recover at arbitrary time instances. We also show that no online algorithm can achieve a bounded competitive ratio if there may be time intervals where no machine is available. In Section 3 we present an online algorithm that constructs schedules with an optimal makespan of $C_{\max}^{OPT}$ if a *lookahead* of one is given, i.e., the algorithm always knows the next point in time when the set of available machines changes. However, the algorithm does not need to know which machines break down or become available. Our algorithm has a running time of $O(an + T^{up})$, where a is the number of time instances where the set of available machines changes and T^{up} is again the time to update the set of available machines. Note that our algorithm has a better running time than Sanlaville's if $a < np_{\max}$, which will be true in practical applications. If $a \geq np_{\max}$, then the set of available machines changes after each time unit. Finally, in Section 4 we give an online algorithm without lookahead that constructs schedules with a nearly optimal makespan of $C_{\max}^{OPT} + \epsilon$, for any $\epsilon > 0$, if at any time at least one machine is available. This implies that not knowing machine availabilities does not really hurt the performance of an algorithm.

2 The performance of online algorithms

First note that if at any time at most one machine is available, an optimal online schedule is trivial to construct. In the following we concentrate on problems with an arbitrary set of machines.

Theorem 1 *No online algorithm can, in general, construct optimal schedules. If there may be time intervals where no machines are available, then no online algorithm can achieve a bounded competitive ratio.*

Proof: Let A be any online algorithm. Initially, at time $t = 0$ only one of m machines is available. We consider n jobs $J_1, \ldots, J_n$, each of which has a processing time of 1 time unit. We asume $n = m$. At time $t = 0$, algorithm A starts processing one job J_{i_0}. Let t' be the first time instance such that A first preempts J_{i_0} or A finishes processing J_{i_0}. At that time t' all machines become available. A's makespan is at least $t' + 1$ because none of the jobs J_i, $i \neq i_0$, has been processed so far. An optimal algorithm will divide the interval from 0 to t' evenly among the n jobs so that its makespan is $C_{\max}^{OPT} = t' + 1 - (t'/n)$. This proves the first part of the theorem. For the proof of the second part we modify the problem instance so that no machine is available during the interval $(C_{\max}^{OPT}, c \cdot C_{\max}^{OPT}]$, for any $c > 1$. The algorithm A cannot finish before $c \cdot C_{\max}^{OPT}$ because it has jobs left at time $C_{\max}^{OPT}$. □

3 Optimal schedules

In this section we give an algorithm that constructs optimal schedules with a makespan of $C_{\max}^{OPT}$. The algorithm is online with a *lookahead of one*, i.e., the algorithm always knows the next point in time when the set of available machines changes. The algorithm does not need to know, however, which machines break down or become available.

Let $J_1, \ldots, J_n$ be the given jobs and let p_i, $1 \leq i \leq n$, denote the processing time of J_i. We assume that p_i is known in advance. Without loss of generality jobs are numbered such that $p_1 \geq p_2 \geq \ldots \geq p_n$. At any time during the scheduling process, r_i denotes the remaining processing time of J_i, $1 \leq i \leq n$. We will show later that the algorithm always maintains the invariant $r_1 \geq r_2 \geq \ldots \geq r_n$.

Starting at time $t = 0$, the algorithm repeatedly schedules time intervals $I = [t, t')$ in which the set of available machines remains the same. The availability changed at t and will next change at time t'. In each interval, the algorithm schedules as much load as possible while minimizing the length of the largest remaining processing time.

More specifically, suppose that the algorithm has already scheduled the interval $[0, t)$ and that the set of available machines changes at t. At time t, using lookahead information, the algorithm determines the next point in time $t' > t$ at which the machine availability changes. Let $\delta = t' - t$ and m^{av} be the number of *av*ailable machines in $I = [t, t')$. Intuitively, the algorithm now tries to determine the largest possible ϵ, $r_1 \geq \epsilon > 0$, such that, for all jobs J_k,

$1 \leq k \leq n$, the remaining processing time in excess to $r_1 - \epsilon$ can be scheduled in I. Thus, at the end of I, all jobs would have a remaining processing time of at most $r_1 - \epsilon$. Figure 1 shows an example. Pictorially, the algorithm determines a vertical line such that the total shaded processing time to the right of this line is equal to total processing capacity available in I. Note that the total processing time in excess to $r_1 - \epsilon$ is

$$\sum_{k=1}^{n} \max\{0, r_k - (r_1 - \epsilon)\}$$

and that the total processing capacity available in I is $m^{av}\delta$. Thus the algorithm computes an ϵ such that $\sum_{k=1}^{n} \max\{0, r_k - (r_1 - \epsilon)\} = m^{av}\delta$.

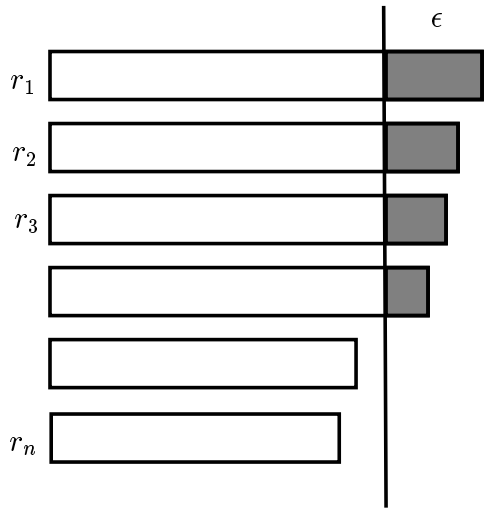

Figure 1: The choice of ϵ

However, the algorithm has to satisfy the constraint that at most δ time units of each job can be scheduled in I. Thus, if $\sum_{k=1}^{n} \max\{0, r_k - (r_1 - \epsilon)\} = m^{av}\delta$ for some $\epsilon > \delta$, then the algorithm cannot schedule ϵ time units of J_1 in I. Only δ time units are permissible.

For this reason, the algorithm first determines a set of jobs that are scheduled for δ time units in I, see lines 6–8 of the code in Figure 4. Suppose that the algorithm has already scheduled δ time units of $J_1, \ldots, J_{i-1}$ in the interval I. Let m_i^{av} be the current number of available machines after these first $i-1$ jobs $J_1, \ldots, J_{i-1}$ have been scheduled. Note that $m_i^{av} = m^{av} - (i-1)$. The total remaining processing capacity in I is equal to $m_i^{av}\delta$. The algorithm also schedules δ time units of J_i in I if the total remaining processing time in excess to $r_i - \delta$ is not sufficient to fill the processing capacity still available and $r_i \geq \delta$. (Formally, if $\sum_{k=i}^{n} \max\{0, r_k - (r_i - \delta)\} < m_i^{av}\delta$ and $r_i \geq \delta$.)

Suppose that the while-loop in lines 6–8 terminates and $i > n$. Then, the algorithm can schedule no more jobs in I. If $i \leq n$, then there are two cases to consider.

(a) $\sum_{k=i}^{n} \max\{0, r_k - (r_i - \delta)\} \geq m_i^{av}\delta$
In this case, the algorithm determines the ϵ, $0 < \epsilon \leq \delta$, such that for all jobs J_k,

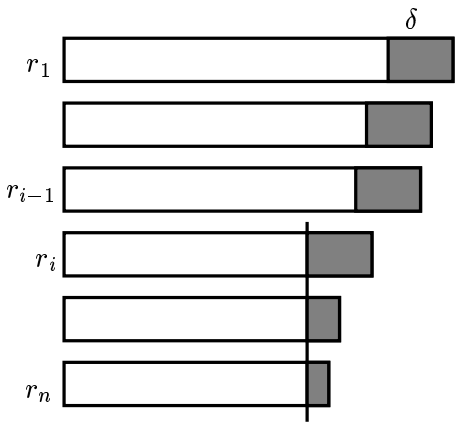

Figure 2: An example of case a)

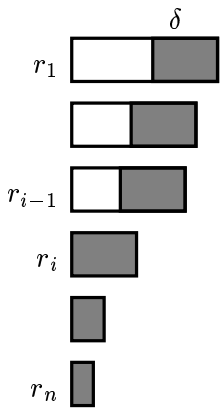

Figure 3: An example of case b)

$i \leq k \leq n$, the total remaining processing time in excess to $r_i - \epsilon$ is exactly equal to $m_i^{av}\delta$, see Figure 2. Each of these jobs is scheduled in I to an extent of $\max\{0, r_k - (r_i - \epsilon)\}$.

(b) $\sum_{k=i}^{n} \max\{0, r_k - (r_i - \delta)\} < m_i^{av}\delta$ and $r_i < \delta$
In this case, the algorithm can schedule the rest of $J_i, \ldots, J_n$, if it exists, in I, see Figure 3.

In each case, the scheduling of the jobs is done using McNaughton's algorithm.

Algorithm Lookahead (LA)

1. $t := 0$;
2. $r_i := p_i$, for $1 \leq i \leq n$;
3. **while** there exist jobs with positive remaining processing time **do**
4. $\quad t' :=$ next point in time when set of available machines changes;
5. $\quad \delta := t' - t$; $i := 1$; $m_1^{av} :=$ number of machines available in $[t, t + \delta)$;
6. $\quad$ **while** $i \leq n$ and $\sum_{k=i}^{n} \max\{0, r_k - (r_i - \delta)\} < m_i^{av}\delta$ and $r_i \geq \delta$ **do**
7. $\quad\quad$ Schedule δ time units of J_i in $[t, t + \delta)$;
8. $\quad\quad$ $r_i := r_i - \delta$; $m_{i+1}^{av} := m_i^{av} - 1$; $i := i + 1$;
9. $\quad$ **if** $i \leq n$ **then**
10. $\quad\quad$ Compute the maximum ϵ, $\epsilon \leq \min\{\delta, r_i\}$, such that $\sum_{k=i}^{n} \max\{0, r_k - (r_i - \epsilon)\} \leq m_i^{av}\delta$;
11. $\quad\quad$ For $k = i, \ldots, n$, schedule $\max\{0, r_k - (r_i - \epsilon)\}$ time units of J_k in $[t, t + \delta)$ using McNaughton's algorithm and set $r_k = \min\{r_k, r_i - \epsilon\}$;
12. $\quad t := t'$;

Figure 4: The online algorithm with a lookahead of one

We analyze the running time of the algorithm and first argue that within an iteration of the outer while-loop, all executions of lines 6–8 take $O(n)$ time. The critical part are the computations of the sums $S_i = \sum_{k=i}^{n} \max\{0, r_k - (r_i - \delta)\}$. Set $S_0 = 0$. We show that S_{i+1}

can be easily derived from S_i. When computing S_i we determine the largest job index l_i such that $r_{l_i} - (r_i - \delta) \geq 0$. We will show below that $r_1 \geq r_2 \geq \ldots \geq r_n$, see Lemma 1. Given l_i, we can easily find l_{i+1} by going through the jobs starting with J_{l_i+1} and find the largest index l_{i+1} such that $r_{l_{i+1}} - (r_{i+1} - \delta) \geq 0$. Then $S_{i+1} = S_i - \delta + (l_i - i)(r_i - r_{i+1}) + \sum_{k=l_i+1}^{l_{i+1}} (r_k - (r_{i+1} - \delta))$. Thus all sums can be computed in $O(n)$ time. Similarly, in line 10, we can compute the desired ϵ in $O(n)$ time. Hence, the scheduling process in each interval $I = [t, t')$ can be done in $O(n)$ time. Thus the total running time of our algorithm is $O(an + T^{up})$, where a is the number of times instances where the set of available machines changes and T^{up} is the time to update the set of available machines. If we represent the set of active machines as a balanced tree, then each machine availability change can be implemented in $O(\log m^{av}_{\max})$ time, where $m^{av}_{\max}$ is the maximum number of machines ever available. Let B denote the total number of machine breakdowns. Then $T^{up} = O(B \log m^{av}_{\max})$.

In the analysis of the algorithm we consider the sequence of intervals in which LA schedules jobs. Within each interval, the set of available machines remains the same. Machine availability only changes at the beginning of an interval.

We first show that the algorithm works correctly. When the algorithm terminates, all jobs have a remaining processing time of zero, i.e. the scheduling process is complete. The condition in line 6 of the algorithm ensures that at most $\delta = t' - t$ time units of each job are scheduled in an interval. The assignment $m^{av}_{i+1} := m^{av}_i - 1$ in line 8 and the constraint $\sum_{k=i}^{n} \max\{0, r_k - (r_i - \epsilon)\} \leq m^{av}_i \delta$ in line 10 ensure that the total amount of processing time scheduled in an interval is not greater than the available processing capacity.

Next we prove two useful lemmas.

Lemma 1 *At the beginning of each interval, $r_1 \geq r_2 \geq \ldots \geq r_n$.*

Proof: The invariant holds at time $t = 0$ because initially $r_k = p_k$, for $1 \leq k \leq n$, and $p_1 \geq p_2 \geq \ldots \geq p_n$. Suppose that $r_1 \geq r_2 \geq \ldots \geq r_n$ holds at the beginning of some interval I. We show that the invariant is also satisfied at the end of I. Let $r'_1, \ldots, r'_n$ denote the remaining processing times at the end of I.

Suppose that while executing the while-loop in lines 6–8, the algorithm schedules δ time units of $J_1, \ldots, J_{i-1}$. The remaining processing time of each of these jobs decreases by δ and thus $r'_1 \geq \ldots \geq r'_{i-1}$. If $i > n$, we are done. Otherwise we have to consider two cases.

(a) $\sum_{k=i}^{n} \max\{0, r_k - (r_i - \delta)\} \geq m^{av}_i \delta$
If $i > 1$, then in the last iteration of the while-loop, the condition in line 6 was satisfied, i.e. $\sum_{k=i-1}^{n} \max\{0, r_k - (r_{i-1} - \delta)\} < m^{av}_{i-1} \delta$, which implies $\sum_{k=i}^{n} \max\{0, r_k - (r_{i-1} - \delta)\} < m^{av}_i \delta$. In line 10, the algorithm chooses an ϵ such that $\sum_{k=i}^{n} \max\{0, r_k - (r_i - \epsilon)\} = m^{av}_i \delta$. Thus, if $i > 1$, $r'_{i-1} = r_{i-1} - \delta > r_i - \epsilon = r'_i$. For any $i \geq 1$, the invariant now follows because $r'_i = \ldots = r'_l$, where l is the largest job index such that $r_l - (r_i - \epsilon) \geq 0$, and $r'_k = r_k$ for $k > l$.

(b) $\sum_{k=i}^{n} \max\{0, r_k - (r_i - \delta)\} < m^{av}_i \delta$ and $r_i < \delta$
In this case, the rest of $J_i, \ldots, J_n$ is scheduled in I, i.e. $r'_i = \ldots = r'_n = 0$ and the invariant holds. □

Now consider any other algorithm A for scheduling $J_1, \ldots, J_n$. In particular, A may be an optimal algorithm that knows the machine breakdowns in advance. At any time consider the sorted sequence $q_1 \geq q_2 \geq \ldots \geq q_n$ of remaining processing times maintained by A. That is, q_i is the i-th value in the sorted sequence, $1 \leq i \leq n$. Note that q_i is not necessarily the remaining processing time of J_i.

Lemma 2 *At the beginning of each interval,* $r_1 \leq q_1$ *and* $\sum_{k=1}^{n} r_k \leq \sum_{k=1}^{n} q_k$.

Proof: We show inductively that at the beginning of each interval

$$\sum_{k=1}^{j} r_k \leq \sum_{k=1}^{j} q_k \quad \text{for} \quad j = 1, \ldots, n. \tag{1}$$

The lemma follows from the special case $j = 1$ and $j = n$. The above inequalities hold at time $t = 0$. Suppose that they hold at the beginning of some interval I. We show that they are also satisfied at the end of I, i.e. at the beginning of the interval following I. Let $r'_1, \ldots, r'_n$ and $q'_1, \ldots, q'_n$ be the remaining processing times at the end of I. Recall that r'_k is the remaining processing time of J_k, $1 \leq k \leq n$. By Lemma 1, $r'_1 \geq \ldots \geq r'_n$. We have $q'_1 \geq \ldots \geq q'_n$ by the definition of the q-values. Note that q_k and q'_k can be the processing times of different jobs. However, $q'_k \leq q_k$ for $1 \leq k \leq n$.

Suppose that in lines 6–8, algorithm LA schedules δ time units of $J_1, \ldots, J_{i-1}$. Then $r'_k = r_k - \delta$, for $k = 1, \ldots, i-1$. We have $q'_k \geq q_k - \delta$, for $1 \leq k \leq n$, because the processing times of jobs decrease by at most δ in I. Thus, inequality (1) holds for $j = 1, \ldots, i-1$. Again, for $i \leq n$, we consider two cases.

(a) $\sum_{k=i}^{n} \max\{0, r_k - (r_i - \delta)\} < m_i^{av}\delta$ and $r_i < \delta$
The algorithm LA schedules the rest of $J_i, \ldots, J_n$ in I so that $r'_i = \ldots = r'_n = 0$. Inequality (1) also holds for $j = i, \ldots, n$.

(b) $\sum_{k=i}^{n} \max\{0, r_k - (r_i - \delta)\} \geq m_i^{av}\delta$
LA computes an ϵ, $0 < \epsilon \leq \delta$, such that $\sum_{k=i}^{n} \max\{0, r_k - (r_i - \epsilon)\} = m_i^{av}\delta$. It reduces the remaining processing times of $J_i, \ldots, J_l$ to $r_i - \epsilon$, where l is the largest job index such that $r_l - (r_i - \epsilon) \geq 0$.

Let m_1^{av} be the number of machines that were initially available in I. Since LA uses all of the available processing capacity, $\sum_{k=1}^{j} r'_k = \sum_{k=1}^{j} r_k - m_1^{av}\delta$ for $j = l, \ldots, n$. Since $\sum_{k=1}^{j} q'_k \geq \sum_{k=1}^{j} q_k - m_1^{av}\delta$ for $j = l, \ldots, n$, inequality (1) holds for $j = l, \ldots, n$. It remains to show that the inequality is also satisfied for $j = i, \ldots, l-1$.

Let $R_1 = \sum_{k=1}^{i-1} r'_k$, $R_2 = \sum_{k=i}^{l} r'_k$ and similarly $Q_1 = \sum_{k=1}^{i-1} q'_k$, $Q_2 = \sum_{k=i}^{l} q'_k$. We have already shown (i) $R_1 \leq Q_1$ and (ii) $R_1 + R_2 \leq Q_1 + Q_2$. Suppose that $Q_1 = R_1 + x$ for some $x \geq 0$. Then (ii) implies $Q_2 + x \geq R_2$. Consider the $l - i + 1$ values $q'_i, \ldots, q'_l$. Since $q'_i \geq \ldots \geq q'_l$, the sum of the first μ values, for any $1 \leq \mu \leq l - i + 1$, is at least $\mu Q_2/(l - i + 1)$. Thus, for any j with $i \leq j \leq l$,

$$\sum_{k=1}^{j} q'_k \quad \geq \quad Q_1 + (j - i + 1)\frac{Q_2}{l - i + 1} \quad = \quad R_1 + x + (j - i + 1)\frac{Q_2}{l - i + 1}$$

$$\geq \quad R_1 + (j-i+1)\frac{Q_2+x}{l-i+1} \;\geq\; R_1 + (j-i+1)\frac{R_2}{l-i+1}$$
$$= \quad \sum_{k=1}^{j} r'_k.$$

The last equation follows because $r_i - \epsilon = r'_i = r'_{i+1} = \ldots = r'_l = R_2/(l-i+1)$. □

Theorem 2 *For any problem instance,* $C^{LA}_{\max} = C^{OPT}_{\max}$.

Proof: Given a set of jobs $J_1, \ldots, J_n$, let $I = [t, t')$ be the last interval in which LA has scheduled jobs, i.e., $t \leq C^{LA}_{\max} \leq t'$. Consider the makespan $C^{OPT}_{\max}$ produced by an optimal offline algorithm. We distinguish two cases.

(1) In the online schedule, the interval from t to $C^{LA}_{\max}$ contains no idle machines
Thus, in the online schedule all machines finish at the same time. Lemma 2 implies that at the beginning of I, the total remaining processing time $\sum_{k=1}^{n} r_k$ of LA is not greater than the total remaining processing time $\sum_{k=1}^{n} q_k$ of OPT. Thus, $C^{OPT}_{\max} \geq C^{LA}_{\max}$.

(2) In the online schedule, the interval from t to $C^{LA}_{\max}$ contains idle machines
Since LA schedules job portions using McNaughton's algorithm, there must exist a job that spans the entire interval from t to $C^{LA}_{\max}$. Thus, at the beginning of I the largest remaining processing time r_1 equals $C^{LA}_{\max} - t$. By Lemma 2, the largest remaining processing time q_1 of OPT is not smaller. Thus OPT cannot finish earlier than LA. □

4 Nearly optimal schedules

In this section we study the problem that an online algorithm has no information about the future machine availabilities. It does not know the next point in time when the set of available machines changes. We present an algorithm that always produces a makespan of $C^{OPT}_{\max} + \epsilon$, for any $\epsilon > 0$. It is assumed that at any time at least one machine is available since otherwise, by Theorem 1, no bounded performance guarantee can be achieved.

We number the jobs to be scheduled such that $p_1 \geq p_2 \geq \ldots \geq p_n$. Given a fixed $\epsilon > 0$, our online algorithm, called *ON(ϵ)*, computes $\delta = \epsilon/n^2$. Starting at time $t = 0$, the algorithm always schedules jobs within the time interval $[t, t+\delta)$. Let m^{av} be the number of machines available at time t. The algorithm determines the m^{av} jobs with the largest remaining processing times (ties are broken arbitrarily) and schedules them on the available machines. If a machine breaks down or becomes available at some time $t + \delta'$, $\delta' < \delta$, then the algorithm preempts the jobs currently being processed and computes a new schedule for the next δ time units from $t + \delta'$ to $t + \delta' + \delta$. Otherwise, if the set of available machines remains the same throughout $[t, t+\delta)$, the algorithm computes a new partial schedule at time $t + \delta$. Let a be the number of time instances where the set of available machines changes. The total number of intervals scheduled by the algorithm is at least a. A formal description

Algorithm Online(ϵ) (ON(ϵ))

1. $t := 0$; $\delta = \epsilon/n^2$;
2. $r_i := p_i$, for $1 \leq i \leq n$;
3. **while** there exist jobs with positive remaining processing time **do**
4. $\quad m^{av}$:= number of machines available at time t;
5. $\quad n^{av}$:= number of jobs with positive remaining processing time;
6. $\quad S$:= set of the $\min\{m^{av}, n^{av}\}$ jobs with the largest remaining processing time;
7. $\quad$ Process the jobs J_i, $i \in S$, on the available machines;
8. $\quad$ **if** machines break down or become available at some time $t + \delta'$, $\delta' < \delta$ **then**
9. $\quad\quad$ Set $r_i := \max\{0, r_i - \delta'\}$ for $i \in S$; $t := t + \delta'$;
10. $\quad$ **else**
11. $\quad\quad$ Set $r_i := \max\{0, r_i - \delta\}$ for $i \in S$; $t := t + \delta$;

Figure 5: The online algorithm *ON(ϵ)*

of the algorithm is given in Figure 5. At any time r_i denotes the remaining processing time of J_i, $1 \leq i \leq n$.

In the scheduling process, the algorithm repeatedly has to find jobs with the largest remaining processing time. If we keep a priority queue of the remaining processing times, each such job can be found in $O(\log n)$ time. Let m_i^{av}, $1 \leq i \leq a$, be the number of machines that are available right after the i-th change; m_0^{av} is the number of machines that are available initially. Let $P = \sum_{i=1}^{n} p_i$. Note that the total number of job portions scheduled by the algorithm is no more than $P/\delta + \sum_{i=0}^{a} m_i^{av}$. This is because at the end of a scheduled job portion, δ time units have been processed or the set of available machines changes. Thus the total running time of the algorithm is $O((Pn^2/\epsilon + \sum_{i=0}^{a} m_i^{av}) \log n + T^{up})$, where T^{up} is the time needed to update the set of available machines. As in the analysis of the algorithm LA we can show that $T^{up} = O(B \log m_{\max}^{av})$, where B is the total number of machine breakdowns and $m_{\max}^{av} = \max_{0 \leq i \leq a} m_i^{av}$. Jobs are only preempted at the end of an interval of length $\delta = \epsilon/n^2$ or when the set of available machines changes. Thus the number of preemptions is no more than $Pn^2/\epsilon + \sum_{i=0}^{a} m_i^{av}$.

For the analysis of the algorithm we partition the time into intervals such that at the beginning of an interval the online algorithm computed a new partial schedule, i.e., it executed lines 4–7. Note that intervals have a length of at most δ and that within each interval the set of available machines remains the same.

The algorithm *ON(ϵ)* does not maintain the property that the remaining processing times $r_1, r_2, \ldots, r_n$ necessarily form a non-increasing sequence (cf. Lemma 1). However, the next lemma shows that if a job J_j has a larger remaining processing time than a job J_i and $i < j$, then the difference is bounded by δ.

Lemma 3 *At the beginning of each interval, for any two jobs J_i and J_j with $i < j$, $r_i \geq r_j - \delta$.*

Proof: The lemma holds at the beginning of the first interval because, initially, $r_1 \geq \ldots \geq r_n$.

Suppose that the lemma holds at the beginning of an interval $I = [t, t + \delta')$, for some $\delta' \leq \delta$. We show that the lemma is also satisfied at the end of I. Let δ_i and δ_j be the number of time units for which J_i and J_j are processed in I. We have $0 \leq \delta_i, \delta_j \leq \delta'$.

If $\delta_j \geq \delta_i$, then there is nothing to show. We study the case $\delta_j < \delta_i$. Let r_k and r'_k, $k \in \{i, j\}$, denote the remaining processing times at the beginning and at the end of I. If $\delta_j = 0$, i.e. only J_i is processed in I, then $r_i \geq r_j$ and thus $r'_i = r_i - \delta_i \geq r_i - \delta \geq r_j - \delta = r'_j - \delta$. Finally, the case $0 < \delta_j < \delta_i$ can only occur if the processing of J_j finishes during I, i.e., $r'_j = 0$. The lemma holds because $r'_i \geq 0$. $\square$

In the following analysis, we have to bound the remaining processing times maintained by *ON(ϵ)* in terms of the remaining processing times maintained by an optimal offline algorithm. In the previous section, when analyzing the algorithm *LA*, we could show that the prefix sum $\sum_{k=1}^{j} r_k$ are bounded by the prefix sum $\sum_{k=1}^{j} q_k$, for $j = 1, \ldots, n$, see (1). Unfortunately, this relation does not hold in the algorithm *ON(ϵ)*. Problems arise if in some interval there exist jobs J_i and J_j with $i < j$ such that J_i is not scheduled but J_j is scheduled in the interval.

For this reason we maintain a sequence of job sets $S_1, \ldots, S_l$, for some $1 \leq l \leq n$, which is a partition of the job sequence $J_1, \ldots, J_n$. Intuitively, a set S_k, $1 \leq k \leq l$, consists of jobs that have "nearly the same" remaining processing time. This will be made precise in Lemma 5. If there are jobs J_i and J_j with $i < j$ such that J_i is not scheduled but J_j is scheduled in some interval, then we merge the sets containing $J_i, \ldots, J_j$. This way we will be able to bound the prefix sums defined by the set $S_1, \ldots S_l$, see Lemma 4 below.

Formally, the sets are maintained as follows. Initially, at time 0, S_i contains J_i, $1 \leq i \leq n$. At the end of each interval I, the sets are updated as follows.

> Let i be the smallest job index such that J_i was not processed in I and let j be the largest job index such that J_j was processed in I. Suppose that $J_i \in S_{k_i}$ and $J_j \in S_{k_j}$. If $k_i < k_j$, then replace $S_{k_i}, S_{k_i+1}, \ldots, S_{k_j}$ by the union of these sets. Renumber the new sequence of sets so that the k-th set in the sequence has index k.

Figure 6 shows an example of the update algorithm for sets. Suppose that in some interval three machines available and jobs J_1, J_3 and J_4 are scheduled for δ time units (the shaded job portions). Job J_2 is the first job not scheduled and J_5 is the last job scheduled in that interval. Thus sets S_1 and S_2 are merged.

Note that, as mentioned above, at any time the sequence of sets forms a partitioning of the jobs $J_1, \ldots, J_n$. The update rule ensures that every set contains a sequence of consecutive jobs with respect to the job numbering. In the following, let n_k denote the number of jobs in S_k, and let $N_k = n_1 + \ldots + n_k$.

At any time let $l_{\max}$ denote the maximum index such that $S_1, \ldots, S_{l_{\max}}$ contain only jobs with positive remaining processing times. If there is no such set, then let $l_{\max} = 0$. Let A be any other scheduling algorithm. In particular, A may be an optimal offline algorithm. At any time consider the sequence of remaining processing times maintained by A, sorted in non-increasing order. Let q_i be the i-th value in this sorted sequence.

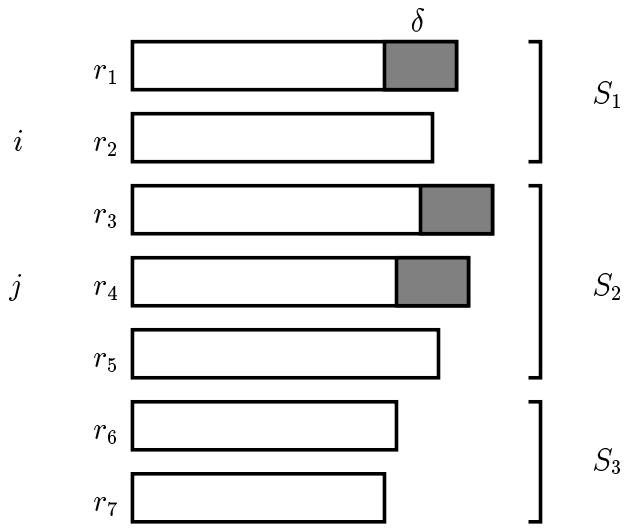

Figure 6: An example of the update rule for sets

Lemma 4 *At the beginning of each interval, for* $k = 1, \ldots, l_{\max}$, $\sum_{i=1}^{N_k} r_i \leq \sum_{i=1}^{N_k} q_i$.

Proof: The lemma holds initially because at time $t = 0$, $r_i = q_i = p_i$ for $1 \leq i \leq n$. Suppose that the lemma holds at the beginning of an interval $I = [t, t + \delta')$, for some $\delta' \leq \delta$. Let $S_1, \ldots, S_l$ and $S'_1, \ldots, S'_{l'}$ be the sequences of job sets at the beginning and at the end of I. Furthermore, let j be the largest index such that all jobs in $S'_1, \ldots, S'_j$ were scheduled in I and still have a positive remaining processing time. These sets were not involved in a merge operation at the end of I and, hence, each S'_k contains the same jobs as S_k, $1 \leq k \leq j$. Since the jobs of these sets have a positive remaining processing time, all of them were scheduled for exactly δ' time units in I. Let r_i, r'_i and q_i, q'_i denote the remaining processing times at the beginning and at the end of I. Since $q'_i \geq q_i - \delta'$, for $1 \leq i \leq n$, we obtain

$$\sum_{i=1}^{N_k} r'_i = \sum_{i=1}^{N_k} (r_i - \delta') \leq \sum_{i=1}^{N_k} (q_i - \delta') \leq \sum_{i=1}^{N_k} q'_i,$$

for $k = 1, \ldots, j$. If $j = l'_{\max}$, then we are done.

Suppose that $j < l'_{\max}$. By the definition of $l'_{\max}$, the set S'_{j+1} does not contain jobs with zero remaining processing time. Also, by the definition of j, S'_{j+1} contains jobs not scheduled in I. The update rule for job sets ensures that S'_{j+1} contains all jobs J_i, $i > N_j$, that were scheduled in I. Let N be the number of jobs in S'_{j+1} scheduled in I. All of these jobs were scheduled for δ' time units because they all have positive remaining processing time. The total number of available machines in I is $N_j + N$ since, otherwise, the algorithm *ON(ϵ)* would have scheduled more jobs of S'_{j+1} in I. Thus any other algorithm cannot process more than $(N_j + N)\delta$ time units in I. We conclude

$$\sum_{i=1}^{N_k} r'_i = \sum_{i=1}^{N_k} r_i - (N_j + N)\delta' \leq \sum_{i=1}^{N_k} q_i - (N_j + N)\delta' \leq \sum_{i=1}^{N_k} q'_i,$$

for $k = j+1, \ldots, l'_{\max}$. $\square$

While $l_{\max} > 0$, the above lemma ensures that an optimal offline algorithm has a total non-zero remaining processing time. When $l_{\max} = 0$, we have to be able to bound the total remaining processing time of *ON(ϵ)*. For this purpose we analyze the difference in remaining times that can occur in a job set S_k.

Lemma 5 *At the beginning of each interval, for every set S_k, $1 \leq k \leq l$, and jobs $J_i, J_j \in S_k$, $|r_i - r_j| \leq (n-1)\delta$.*

The bound given in the above lemma is an overestimate, which is sufficient for the rest of the analysis. However, there exist problem instances such that $|r_i - r_j| \geq (n/2)\delta$.

Proof: We prove inductively that at the beginning of each interval, for every set S_k and jobs $J_i, J_j \in S_k$,

$$|r_i - r_j| \leq (n_k - 1)\delta. \tag{2}$$

This holds initially because at time $t = 0$, every set contains exactly one job. Consider an interval $I = [t, t+\delta')$, for some $\delta' \leq \delta$, and suppose (2) holds at the beginning of I.

We first show that (2) is maintained while jobs are processed in I and before the update rule for the sets is applied. Given a set S_k, let $J_i, J_j \in S_k$ be any two jobs with $i < j$. Let r_i, r'_i and r_j, r'_j be the remaining processing times at the beginning and at the end of I. If $r'_i \leq r'_j$, then by Lemma 3, $|r'_i - r'_j| = r'_j - r'_i \leq \delta$.

If $r'_i > r'_j$, we have to consider several cases. If none of the two jobs was processed in I or if both jobs were processed for δ' time units, then there is nothing to show. Otherwise, let δ_i and δ_j be the number of time units for which J_i and J_j are processed in I. If only J_j is processed in I, then $r_j \geq r_i$ and thus $|r'_i - r'_j| = r'_i - r'_j = r_i - (r_j - \delta_j) \leq \delta_j \leq \delta$. The case that both J_i and J_j are scheduled in I, but J_j is processed for a longer period, cannot occur. This would imply that the processing of J_i is complete, i.e. $r'_i = 0$, which contradicts $r'_i > r'_j$. Finally suppose that J_i is processed as least as long as J_j in I, i.e. $0 \leq \delta_j \leq \delta_i$. Then $|r'_i - r'_j| = r_i - \delta_i - (r_j - \delta_j) = r_i - r_j + \delta_j - \delta_i \leq r_i - r_j \leq (n_k - 1)\delta$. Inequality (2) is satisfied.

We now study the effect when the set update rule is applied at the end of I. Suppose that a sequence of sets $S_{k_1}, \ldots, S_{k_2}$ is merged. Let $J_i \in S_{k_1}$ be a job not scheduled in I and let $J_j \in S_{k_2}$ be the job with the largest index scheduled in I. Let $J_{\max}$ be the job in $S_{k_1}, \ldots, S_{k_2}$ with the largest remaining processing time at time $t+\delta'$ and let $J_{\min}$ be the job in $S_{k_1}, \ldots, S_{k_2}$ with the smallest remaining processing time. We will show $|r'_{\max} - r'_{\min}| \leq (n_{k_1} + n_{k_2} - 1)\delta$. This completes the proof because the newly merged set contains $\sum_{k=k_1}^{k_2} n_k \geq n_{k_1} + n_{k_2}$ jobs. We have

$$|r'_{\max} - r'_{\min}| = r'_{\max} - r'_{\min} = (r'_{\max} - r'_i) + (r'_i - r'_j) + (r'_j - r'_{\min}).$$

If $J_{\max} \in S_{k_1}$, then $r'_{\max} - r'_i \leq (n_{k_1} - 1)\delta$. If $J_{\max} \notin S_{k_1}$, then $r'_{\max} - r'_i \leq \delta$ by Lemma 3 because $J_{\max}$ has a higher index than J_i. In any case $r'_{\max} - r'_i \leq (n_{k_1} - 1)\delta$. Similarly, if $J_{\min} \in S_{k_2}$, then $r'_j - r'_{\min} \leq (n_{k_2} - 1)\delta$. If $J_{\min} \notin S_{k_2}$, then $r'_j - r'_{\min} \leq \delta$ by Lemma 3. In any case $r'_j - r'_{\min} \leq (n_{k_2} - 1)\delta$. Since J_i was not scheduled in I but J_j was scheduled, $r_j \geq r_i$.

Job J_j was scheduled for at most δ time units, which implies $r'_i = r_i \leq r_j \leq r'_j + \delta$ and hence $r'_i - r'_j \leq \delta$. In summary we obtain

$$\begin{aligned} |r'_{\max} - r'_{\min}| &= (r'_{\max} - r'_i) + (r'_i - r'_j) + (r'_j - r'_{\min}) \\ &\leq (n_{k_1} - 1)\delta + \delta + (n_{k_2} - 1)\delta \\ &= (n_{k_1} + n_{k_2} - 1)\delta. \ \square \end{aligned}$$

Theorem 3 *For any fixed $\epsilon > 0$ and any problem instance, $C_{\max}^{ON(\epsilon)} \leq C_{\max}^{OPT} + \epsilon$.*

Proof: Let $I = [t, t + \delta')$, $\delta' \leq \delta$, be the last interval such that $l_{\max} > 0$ at the beginning of I. Consider the total remaining processing time of the jobs in $S_1, \ldots, S_{l_{\max}}$ at time t. By Lemma 4, the value of *ON(ϵ)* is not larger than the value of an optimal offline algorithm. Thus $C_{\max}^{OPT} \geq t$. We analyse *ON(ϵ)*'s makespan. At time $t+\delta'$, S_1 contains a job J_i with zero remaining processing time. By Lemma 5, all jobs belonging to the first set have a remaining processing time of at most $(n-1)\delta$. All jobs not belonging to the first set have a higher index than J_i and, by Lemma 3, they have a remaining processing time of at most δ. Thus at time $t + \delta'$, we are left with at most $n - 1$ jobs having a remaining processing time of at most $(n-1)\delta$, i.e., the total remaining processing time of *ON(ϵ)* is at most $(n-1)^2\delta$. Since at any time at least one machine is available $C_{\max}^{ON(\epsilon)} \leq t+\delta'+(n-1)^2\delta \leq C_{\max}^{OPT}+n^2\delta \leq C_{\max}^{OPT}+\epsilon$. $\square$

Acknowledgment

We thank Oliver Braun for many interesting discussions. Moreover, we thank two anonymous referees for their helpful comments improving the presentation of the paper.

References

[1] B. Kalyanasundaram and K.P. Pruhs. Fault-tolerant scheduling. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pages 115–124, 1994.

[2] B. Kalyanasundaram and K.P. Pruhs. Fault-tolerant real-time scheduling. In *Proc. 5th Annual European Symposium on Algorithms (ESA)*, Springer Lecture Notes in Computer Science, 1997.

[3] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6:1-12, 1959.

[4] E. Sanlaville. Nearly on line scheduling of preemptive independent tasks. *Discrete Applied Mathematics*, 57:229–241, 1995.

[5] E. Sanlaville. Private communication, 1998.

[6] E. Sanlaville and G. Schmidt. Machine scheduling with availablity constraints. *Acta Informatica*, 35:795–811, 1998.

[7] G. Schmidt. Scheduling on semi-identical processors. *Z. Oper. Res.*, 28:153–162, 1984.

[8] G. Schmidt. Scheduling independent tasks with deadlines on semi-identical processors. *J. Oper. Res. Soc.*, 39:271–277, 1988.

[9] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.