A Framework for Scheduling with Online Availability

Florian Diedrich[⋆,⋆⋆] and Ulrich M. Schwarz[⋆⋆]

Institut für Informatik,
Christian-Albrechts-Universität zu Kiel,
Olshausenstr. 40, 24098 Kiel, Germany
{fdi,ums}@informatik.uni-kiel.de

Abstract. With the increasing popularity of large-scale distributed computing networks, a new aspect has to be considered for scheduling problems: machines may not be available permanently, but may be withdrawn and reappear later. We give several results for completion time based objectives: 1. we show that scheduling independent jobs on identical machines with online failures to minimize the sum of completion times is $(8/7 - \epsilon)$-inapproximable, 2. we give a nontrivial sufficient condition on machine failure under which the SRPT (shortest remaining processing time) heuristic yields optimal results for this setting, and 3. we present meta-algorithms that convert approximation algorithms for offline scheduling problems with completion time based objective on identical machines to approximation algorithms for the corresponding preemptive online problem on identical machines with discrete or continuous time. Interestingly, the expected approximation rate becomes worse by a factor that only depends on the probability of unavailability.

1 Introduction

Since the advent of massive parallel scheduling, machine unavailability has become a major issue. Machines can be damaged and thus are not operational until some maintainance is undertaken, or idle time is donated by machines. In either case, it is a realistic assumption that the time of availability of the individual machines can neither be controlled nor foreseen by the scheduler.

Related problems and previous results. In classical scheduling, dynamic machine unavailability has at best played a minor role; however, unreliable machines have been considered as far back as 1975 [1] in the offline setting. Semi-online adversarial variants of the makespan problem were studied by Sanlaville [2] and Albers & Schmidt [3]. In the semi-online setting, the next point in time when machine

⋆ Research supported in part by a grant "DAAD Doktorandenstipendium" of the German Academic Exchange Service; part of this work was done while visiting the LIG, Grenoble.

⋆⋆ Research supported by EU research project AEOLUS, Algorithmic Principles for Building Efficient Overlay Computers, EU contract number 015964.

availability may change is known. The discrete time step setting considered here is a special case (i.e. we assume that at time $t+1$, availability changes) that is closely linked to the unit execution time model. Sanlaville & Liu [4] have shown that *longest remaining processing time* (LRPT) is an optimal strategy for minimizing the makespan even if there are certain forms of precedence constraints on the jobs.

Albers & Schmidt [3] also give results on the true online setting which are obtained by imposing a "guessed" discretization of time.

The general notion of solving an online problem by re-using offline solutions was used by Hall et al. [5] and earlier by Shmoys et al. [6], where $\sum w_j C_j$ and $\min \max C_j$ objectives, respectively, with online job arrivals were approximated using corresponding or related offline algorithms.

Applications. We focus mainly on a setting where there is a large number of individual machines which are mainly used for other computational purposes and donate their idle periods to perform fractions of large computational tasks. The owner of these tasks have no control of the specific times of availability; there even might be no guarantee of availability at all. Our model provides a formal framework to deal with various objective functions in such a situation.

New results. We first study a setting of discrete time steps where the jobs are known in advance and availability is revealed on-line and give three main results: we prove $(8/7-\epsilon)$-inapproximability for any online algorithm that tries to minimize the average completion time; we show that the *shortest remaining processing time first* heuristic SRPT solves this problem optimally if the availability pattern is increasing zig-zag; and finally, we present a meta-algorithm for a stochastic failure model that uses offline approximations and incurs an additional approximation factor that depends in an intuitive way on the failure probability. Our approach holds for any offline preemptive algorithm that approximates the makespan or the (possibly weighted) sum of completion times, even in the presence of release dates and in-forest precedence constraints, and slightly weaker results are obtained for general precedence constraints. We also show how our results can be adapted to a semi-online model of time, i.e. breakdowns and preemptions can occur at any time, but the time of the next breakdown is known in advance.

In our probabilistic model of machine unavailability, for every time step t each machine is unavailable with probability $f \in [0,1)$; this can also be seen as a failure probability of f where there is a probability of $1-f$ that the machine will be available again at the next time. Notice that in the setting studied in [7], no such probability assumptions are made; here machines just are statically available or unavailable.

2 Definitions

We will consider the problem of scheduling n jobs $J_1, \ldots, J_n$, where a job J_i has processing time p_i which is known *a priori*. We denote the completion time of J_i

in schedule σ with $C_i(\sigma)$ and drop the schedule where it is clear from context. We denote the number of machines available for a given time t as $m(t)$ and the total number of machines $m = \max_{t\in\mathbb{N}} m(t)$. The SRPT algorithm simply schedules the jobs of shortest remaining processing time at any point, preempting running jobs if necessary.

3 Lower bounds

In the following, we assume that at any time, at least one machine is available; otherwise, no algorithm can have bounded competitive ratio unless it always minimizes the makespan as well as the objective function.

Theorem 1. *For* $\mathrm{P}m, fail|pmtn|\sum C_j$*, no online algorithm has competitive ratio* $\alpha < 8/7$*.*

Proof. Consider instance I with $m = 3$ and 3 jobs, where $p_1 = 1$ and $p_2 = p_3 = 2$. Let ALG be any online algorithm for our problem. The time horizon given online will start with $m(1) = 1$. We distinguish two cases depending on the behaviour of the algorithm. In *Case 1*, ALG starts J_1 at time 1, resulting in $C_1(\sigma_1) = 1$. Then the time horizon is continued by $m(2) = 3$ and finally $m(3) = m(4) = 1$. It is easy to see that the best sum of completion times the algorithm can attain is 8. However, starting J_2 at time 1 yields a schedule with a value of 7.

In *Case 2*, ALG schedules J_2 at time 1; the same argument works if J_3 is scheduled here. The time horizon is continued by $m(2) = m(3) = 2$. Enumerating the cases shows that the algorithm can only get $\sum C_j \geq 8$, however, by starting J_1 at time 1, an optimal schedule gets $\sum C_j = 7$. □

Theorem 2. *For* $\mathrm{P}m, fail|pmtn|\sum C_j$*, the competitive ratio of SRPT is* $\Omega(m)$*.*

Proof. For $m \in \mathbb{N}$, m even, consider m machines and m small jobs with $p_1 = \cdots = p_m = 1$ and $m/2$ large jobs with $p_{m+1} = \cdots = p_{m+m/2} = 2$. We set $m(1) = m(2) = m$ and $m(t) = 1$ for every $t > 2$.

SRPT generates a schedule σ_2 by starting the m small jobs at time 1 resulting in $C_j(\sigma_2) = 1$ for each $j \in \{1, \ldots, m\}$. At time 2 all of the $m/2$ large jobs are started; however, they cannot be finished at time 2 but as time proceeds, each of them gets executed in a successive time step. This means that $C_{m+j}(\sigma_2) = 2 + j$ holds for each $j \in \{1, \ldots, m/2\}$. In total, we obtain $\sum C_j = m + \sum_{j=1}^{m/2}(2 + j) = \Omega(m^2)$.

A better schedule will start all long jobs at time 1 and finishes all jobs by time 2, for $\sum C_j \leq 3m$. □

4 SRPT for special availability patterns

Throughout this section, we assume the following availability pattern which has been previously studied for min-max objectives [7]:

Definition 1. *Let $m : \mathbb{N} \to \mathbb{N}$ the machine availability function; m forms an* increasing zig-zag pattern *iff the following condition holds:*

$$\forall t \in \mathbb{N} : m(t) \geq \max_{t' \leq t} m(t') - 1 \,.$$

Intuitively, we may imagine that machines may join at any time and that only one of the machines is unreliable.

Lemma 1. *$p_i < p_j$ implies $C_i \leq C_j$ for each $i, j \in \{1, \ldots, n\}$ in some suitable schedule σ that minimizes $\sum C_j$.*

Proof. Fix a schedule σ that minimizes $\sum C_j$. Let $i, j \in \{1, \ldots, n\}$ with $p_i < p_j$ but $C_i > C_j$. Let I_i, I_j be the sets of times in which J_i, J_j are executed in σ, respectively. We have $0 < |I_i \setminus I_j| < |I_j \setminus I_i|$ since $p_i < p_j$, $C_i > C_j$. Let $g : I_i \setminus I_j \to I_j \setminus I_i$ be an injective mapping; construct a schedule σ' from σ in the following way: for all $t \in I_i \setminus I_j$ exchange the execution of J_i at time t with the execution of J_j at time $g(t)$. Then we have $C_k(\sigma) = C_k(\sigma')$ for every $k \neq i, j$, furthermore $C_i(\sigma) = C_j(\sigma')$ and $C_j(\sigma) \geq C_i(\sigma')$. Iterating the construction yields the claim. □

Theorem 3. *SRPT is an optimal algorithm if machine availabilities form an increasing zig-zag pattern.*

Proof. Assume a counterexample $J_1, \ldots, J_n$, m such that $\sum_{j=1}^{n} p_j$ is minimal. Fix an optimal schedule σ_{OPT} and an SRPT schedule σ_{ALG} such that the set D of jobs that run at time 1 in only one of $\sigma_{\mathrm{OPT}}, \sigma_{\mathrm{ALG}}$ is of minimal size.

If $|D| = 0$, then σ_{OPT} and σ_{ALG} coincide at time 1 up to permutation of machines. Denote with I the set of jobs running at time 1. By definition of SRPT, $I \neq \emptyset$. We define a new instance by setting

$$\forall j = 1, \ldots, n : p'_j := \begin{cases} p_j - 1, & J_j \in I, \\ p_j, & J_j \notin I \end{cases}$$
$$\forall t \in \mathbb{N} : m'(t) := m(t+1) \,.$$

It is obvious that $\sum p'_j < \sum p_j$, and so it would be a smaller counterexample. Hence, $D \neq \emptyset$.

We will now argue that there must be some job run by σ_{OPT} that is not run by σ_{ALG} at time 1 and then show that we can exchange these jobs in σ_{OPT} without increasing the objective function value, leading to a counterexample of smaller $|D|$.

Assume that all jobs run by σ_{OPT} also run in σ_{ALG}. Since $|D| > 0$, there is some job in σ_{ALG} that is not in σ_{OPT}, hence σ_{OPT} contains an idle machine. Hence, all n available jobs must run in σ_{OPT} at time 1 by optimality, a contradiction to $|D| > 0$.

Thus there is a job J_j run by σ_{OPT} which is not run by σ_{ALG}. Since not all n jobs can run in σ_{ALG} at time 1 and SRPT is greedy, there must be a different

job J_i which is run in σ_{ALG}, but not in σ_{OPT}. By definition of SRPT, we know $p_i < p_j$, and $C_i(\sigma_{\mathrm{OPT}}) \leq C_j(\sigma_{\mathrm{OPT}})$ by Lemma 1.

We will now show that it is always possible to modify σ_{OPT} to execute J_i at time 1 instead of J_j. Preferring J_i will decrease its completion time by at least 1, so we need to show that the total sum of completion times of the other jobs is increased by at most 1.

Case 1: if J_j does not run at time C_i in σ_{OPT}, we have $C_j > C_i$ and we can execute J_i at time 1 and J_j at time C_i. This does not increase the completion time C_j, and any other job's completion time remains unchanged.

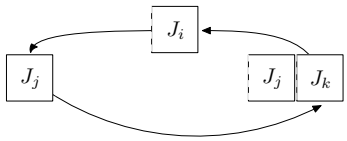

Fig. 1: Case 2 in the proof of Theorem 3

Case 2: The following construction is sketched in Fig. 1. J_j runs at time C_i. We will execute J_i at time 1 and J_j at time $C_j + 1$ for a total change of $\sum C_j$ of at most 0. This can trivially be done if there is an idle machine in σ_{OPT} at time $C_j + 1$. Otherwise, there are $m(C_j + 1)$ jobs running at that time. We still have an idle machine at time C_i, freed up by moving J_i to time 1, and want to displace one of the $m(C_j + 1)$ jobs into this space. We note that we may not choose jobs that are already running at time C_i. There are at most $m(C_i) - 2$ such jobs, since we know J_i and J_j are running at that time. By the increasing zig-zag condition and $C_i \leq C_j < C_j + 1$, we know that

$$m(C_j + 1) \geq m(C_i) - 1 > m(C_i) - 2\,,$$

so at least one job, say J_k, is not excluded. Since no part of J_k is delayed, C_k does not increase. □

5 Algorithm MIMIC

The basic idea of algorithm MIMIC is to use an offline approximation for reliable machines and re-use this given schedule as far as possible. More precisely, let us assume that we already have an α-approximate schedule σ for the offline case for an objective in $\{\sum w_j C_j, \sum C_j, C_{\max}\}$. We will first convert the schedule into a queue Q in the following way; we note that this is for expository reasons and not needed in the implementation.

> For any time t and any machine $i \in \{1, \ldots, m\}$, the job running at time t on machine i in schedule σ is at position $(t-1)m + i$ in the queue. (1)

Note that this means "idle" positions may occur in the queue; this is not exploited. We can now use the queue in our online scheduling algorithm in Fig. 2.

Setup: calculate Q.
Online Execution: if at time t, $m(t)$ machines are available, preempt all currently running jobs, remove the first $m(t)$ entries from Q and start the indicated jobs. Terminate when Q becomes empty.

Fig. 2: Algorithm MIMIC for independent jobs

Remark 1. In the generated schedule, no job runs in parallel to itself.

Proof. We assume w.l.o.g. that there are no redundant preemptions in the offline schedule σ, i.e. if a job j runs at time t as well as at time $t+1$, it remains on the same machine. Hence, two entries in the queue corresponding to the same job must be at least m positions apart. Since at no time in the online schedule, more than m machines are available, no two entries of the same job can be eligible simultaneously. □

We now take a different view upon machine failure: instead of imagining failed machines, we consider that "failure blocks" are inserted into the queue. Since there is a one-to-one correspondence of machine/time positions and queue positions given by (1), this is equivalent to machine failures. We recall an elementary probabilistic fact:

Remark 2 (Expected run length). In our setting (machines failing for a time step independently with probability f), the expected number of failure blocks in front of each non-failure block is $f/(1-f)$.

We can now bound how long the expected completion of a single job is delayed in the online schedule σ':

Lemma 2. *For any job j, we have $E[C_j(\sigma')] \leq \frac{1}{1-f} C_j(\sigma) + 1$.*

Proof. We note that since there are always m machines in the offline setting, there cannot be any blocks corresponding to j in the queue after position $mC_j(\sigma)$ before failure blocks are inserted. This means that after random insertion of the failure blocks, the expected position of the last block of job j is at most $\big(1 + f/(1-f)\big) mC_j(\sigma)$. In light of (1), this yields

$$E[C_j(\sigma')] = \lceil \frac{1}{m}(mC_j(\sigma)\frac{1}{1-f}) \rceil \leq \frac{1}{1-f} C_j(\sigma) + 1\,,$$

which proves the claim. □

Theorem 4. *MIMIC has asymptotic approximation ratio $1/(1-f)$ for independent unweighted jobs and $(1+\epsilon)/(1-f)$ for sum of weighted completion times with release dates.*

This is achieved by exploiting known offline results for different settings (cf. Tab. 1). We should note in particular that since machine failure at most delays a job, our model is applicable to settings with non-zero release dates. We list the result of Kawaguchi & Kyan mainly because it is obtained by a very simple *largest ratio first* heuristic, as opposed to the more sophisticated methods of Afrati et al. [8], which gives it a very low computational complexity.

Table 1: Selection of known offline results that can be used by MIMIC.

Setting	Source		ax. ratio
$P\|pmtn\|\sum C_j$	McNaughton	[9]	1
$P\|\|\sum w_j C_j$	Kawaguchi and Kyan	[10]	$(1+\sqrt{2})/2$
$P\|r_j, pmtn\|\sum w_j C_j$	Afrati et al.	[8]	PTAS
$P\|r_j, prec, pmtn\|\sum w_j C_j$	Hall et al.	[5]	3

5.1 Handling precedence constraints

We note that our results stated so far cannot be simply used if there are general precedence constraints, as the following example shows:

Example 1. Consider four jobs $J_1, \ldots, J_4$ of unit execution time such that $J_1, J_2 \prec J_3, J_4$. The queue $J_1 J_2 J_3 J_4$ corresponds to an optimal offline schedule. If a failure occurs during the first time step, we have $C_2 = 2$ and MIMIC schedules one of J_3, J_4 in parallel to J_2.

The main problem is that jobs J_2 and J_3 have a distance of $1 < m = 2$ in the queue, so they may be scheduled for the same time step online even though there is a precedence constraint on them. Conversely, if the distance is at least m, they are never scheduled for the same time step.

Since our setting allows free migration of a job from one machine to another, we can sometimes avoid this situation: if the precedence constraints form an *in-forest*, i.e. every job has at most one direct successor, we can rearrange the jobs in the following way: if at a time t, a job J_0 is first started, and $J_1, \ldots, J_k, k \geq 1$ are those of J_0's direct predecessors that run at time $t-1$, w.l.o.g. on machines $1, \ldots, k$, we assign J_0 to machine k. This ensures that the distance in the queue from J_0 to J_k and hence also to $J_1, \ldots, J_{k-1}$ is at least m.

If we have general precedence constraints, we cannot guarantee that all jobs are sufficiently segregated from their predecessors, as seen above. We can, however, use a parameterized modification to extend results to the case of general precedence constraints and only incur an additional factor of $1+\epsilon$: fix $k \in \mathbb{N}$ such that $k^{-1} \leq \epsilon$. We will use time steps of granularity k^{-1} in the online schedule.

For the offline instance, we first scale up all execution times by k, essentially shifting them to our new time units, and then increase all execution times by 1.

Setup: Calculate offline schedule σ_{offline}; $t_{\text{offline}} := 0$
Online Execution: if $m(t)$ machines are available during the interval $[t, t+\delta)$:

- Set $\delta_{\text{offline}} := \min\{m(t)\delta/m,\ \min\{C_j(\sigma_{\text{offline}}) - t_{\text{offline}} | t_{\text{offline}} \leq C_j(\sigma_{\text{offline}})\}\}$.
- Set $\delta_{\text{online}} := m\delta_{\text{offline}}/m(t)$.
- Schedule all job fractions that run in the interval $[t_{\text{offline}}, t_{\text{offline}} + \delta_{\text{offline}})$ in σ_{offline} in the online interval $[t, t+\delta_{\text{online}})$ unsing McNaughton's wrap-around rule.
- Set $t_{\text{offline}} := t_{\text{offline}} + \delta_{\text{offline}}$ and proceed to time $t + \delta_{\text{online}}$.

Fig. 3: Algorithm MIMIC′

Since all scaled execution times were at least k to begin with, this gives at most an additional factor of $1 + k^{-1} \leq 1 + \epsilon$ for the objective function value. In the queue, we leave the additional time slot empty. This forces the queue distance of jobs which are precedence-constrained to be at least m, hence, the generated schedule is valid.

6 Continuous time and the semi-online model

In this section, we will adapt the idea of algorithm MIMIC—reusing an offline approximation—to the more general semi-online setting by methods similar to Prasanna & Musicus' continuous analysis [11]. In the semi-online setting, changes of machine availability and preemptions may occur at any time whatsoever, however, we know in advance the next point in time when a change of machine availability will take place. We can use this knowledge to better convert an offline schedule into an online schedule, using the algorithm MIMIC′ in Fig. 3: during each interval of constant machine availability, we calculate the area $m(t)\delta$ we can schedule. This area will be used up in time $m(t)\delta/m$ in the offline schedule. We take the job fractions as executed in the offline schedule and schedule them online with McNaughton's wrap-around rule [9]. Precedence constraints can be handled by suitable insertion of artificial interruptions.

Since at time $C_j(\sigma_{\text{offline}})$, a total area of $mC_j(\sigma_{\text{offline}})$ is completed, we have the following bound on the online completion times $C_j(\sigma_{\text{online}})$:

$$\int_0^{C_j(\sigma_{\text{online}})} m(t)dt \leq mC_j(\sigma_{\text{offline}})\,. \tag{2}$$

If we set $\forall t : E[m(t)] = (1-f)m$ to approximate our independent failure setting above, equation (2) simplifies to $C_j(\sigma)(1-f)m \leq mC_j(\sigma_{\text{offline}})$, which again yields a $1/(1-f)$-approximation as in Lemma 2, thus we obtain the following result.

Theorem 5. *Algorithm MIMIC' non-asymptotically matches the approximation rates of MIMIC for the continuous semi-online model.*

7 Conclusion

In this paper we have presented a simple yet general framework permitting the transfer of results from offline scheduling settings to natural preemptive online extensions. We have studied the min-sum objectives $\sum C_j$ and $\sum w_j C_j$; we have also considered the behaviour of $C_{\max}$, which permits the transfer of bicriteria results. We remark that algorithm MIMIC permits a fast and straightforward implementation which indicates its value as a heuristic for practice where real-time data processing is important; this holds in particular when the underlying offline scheduler has low runtime complexity.

Acknowledgements. The authors thank Jihuan Ding for many fruitful discussions on lower bounds, and the anonymous referees for their valuable comments which helped improve the quality of the exposition.

References

1. Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**(3) (1975) 384–393
2. Sanlaville, E.: Nearly on line scheduling of preemptive independent tasks. Discrete Applied Mathematics **57**(2-3) (1995) 229–241
3. Albers, S., Schmidt, G.: Scheduling with unexpected machine breakdowns. Discrete Applied Mathematics **110**(2-3) (2001) 85–99
4. Liu, Z., Sanlaville, E.: Preemptive scheduling with variable profile, precedence constraints and due dates. Discrete Applied Mathematics **58**(3) (1995) 253–280
5. Hall, L.A., Shmoys, D.B., Wein, J.: Scheduling to minimize average completion time: Off-line and on-line algorithms. In: Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms. (1996) 142–151
6. Shmoys, D.B., Wein, J., Williamson, D.P.: Scheduling parallel machines on-line. SIAM J. Comput. **24**(6) (1995) 1313–1331
7. Sanlaville, E., Schmidt, G.: Machine scheduling with availability constraints. Acta Informatica **35**(9) (1998) 795–811
8. Afrati, F.N., Bampis, E., Chekuri, C., Karger, D.R., Kenyon, C., Khanna, S., Milis, I., Queyranne, M., Skutella, M., Stein, C., Sviridenko, M.: Approximation schemes for minimizing average weighted completion time with release dates. In: Proceedings of FOCS '99. (1999) 32–44
9. McNaughton, R.: Scheduling with deadlines and loss functions. Mgt. Science **6** (1959) 1–12
10. Kawaguchi, T., Kyan, S.: Worst case bound of an LRF schedule for the mean weighted flow-time problem. SIAM Journal on Computation **15**(4) (1986) 1119–1129
11. Prasanna, G.N.S., Musicus, B.R.: The optimal control approach to generalized multiprocesor scheduling. Algorithmica **15** (1996) 17–49