Scheduling on unreliable machines

AEOLUS Workshop on Scheduling

Florian Diedrich   Ulrich Schwarz

Christian-Albrechts-Universität zu Kiel

March 8, 2007

Overview

The setting

Given	a set J of preemptible jobs, a set M of machines precedence constraints, release times
Problem	machines may "fail" online: only $M(t) \subseteq M$ will contribute at time t.
Goal	minimize makespan or sum of completion times

A simple example

Example

Identical machines, failure probability $0 \leq f < 1$ constant.
Minimize $\sum C_j$.

Main Idea

This would be easy if machines couldn't fail. (SRPT)
Use as much information from such a solution as possible.

Two steps:

1. Build optimal offline schedule for all m machines.
2. Online:
   - try to clear up backlog
   - do as the offline schedule says

A simple example (cont.)

Formally:

Algorithm MIMIC

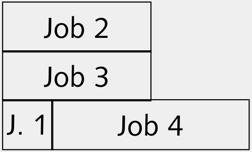
1. Calculate offline schedule
2. Build queue: job on machine i in interval t goes into position $tm + i$.
3. m' machines available online:
   schedule first m' jobs from queue.

Remark

Step 1, 2 need not be explicit.

A simple example (cont.)

Offline



Queue

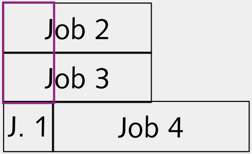
2	3	1	2	3	4	2	3	4			4			4

Online

✘

A simple example (cont.)

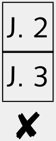
Offline



Queue

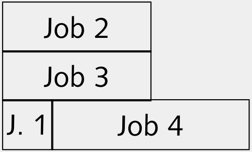
2	3	1	2	3	4	2	3	4			4			4

Online



A simple example (cont.)

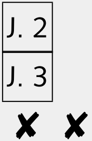
Offline



Queue

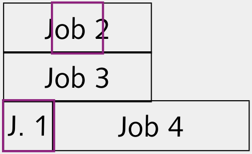
2	3	1	2	3	4	2	3	4			4			4

Online



A simple example (cont.)

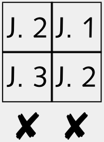
Offline



Queue

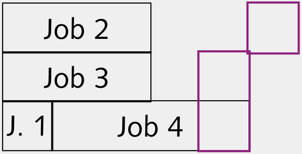
2	3	1	2	3	4	2	3	4			4			4

Online



A simple example (cont.)

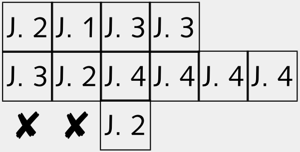
Offline



Queue

2	3	1	2	3	4	2	3	4			4			4	

Online



A simple example (cont.)

Example

Identical machines, failure probability $0 \leq f < 1$ constant.

Lemma

Any job's expected completion time is delayed from C_J to $\frac{1}{1-f} C_J + 1$.

Theorem

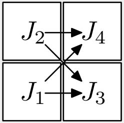
MIMIC is a $\frac{1}{1-f}$-approximation for $C_{\max}, \sum C_j, \sum w_j C_j$. (Asymptotically.)

More structured instances

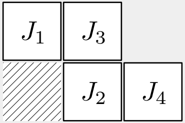
Release dates results hold trivially.

Precedences work only sometimes:

Offline



Online?



- In-trees always work
- Otherwise: may create one idle step per job

A continuous model of time

Up to now: things happen in discrete time steps.

Semi-online setting

- preemption at any time
- we know the next time the set of available machines changes (*event*).

Known:

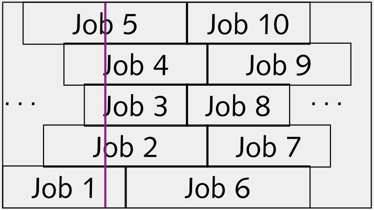
Theorem (Albers/Schmidt 99)

There is an optimal polynomial-time online algorithm for the $C_{\max}$ objective and independent jobs on identical machines.

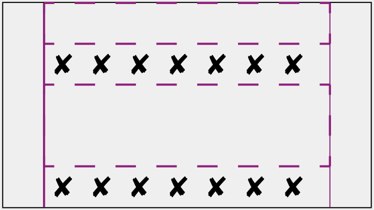
MIMIC in the continuous case

- Calculate online area until next event.
- Consider corresponding offline interval.
- Insert artificial events for job termination.
- Schedule jobs with McNaughton's wraparound rule.

Offline



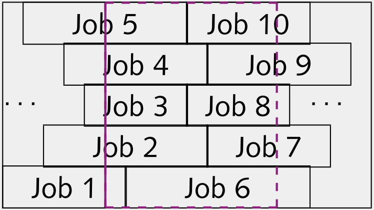
Online



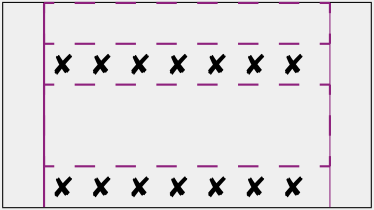
MIMIC in the continuous case

- Calculate online area until next event.
- Consider corresponding offline interval.
- Insert artificial events for job termination.
- Schedule jobs with McNaughton's wraparound rule.

Offline



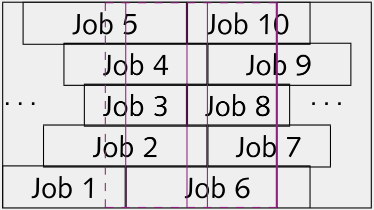
Online



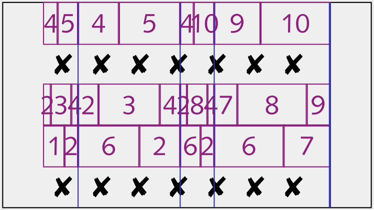
MIMIC in the continuous case

- Calculate online area until next event.
- Consider corresponding offline interval.
- Insert artificial events for job termination.
- Schedule jobs with McNaughton's wraparound rule.

Offline



Online



Non-identical machines: it's not that easy

When machines are not identical, migration can be harmful:

Offline

(Speed 4)	Job 1
(Speed 1)	Job 2
(Speed 1)	Job 3

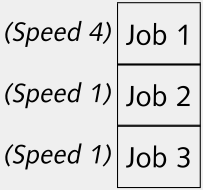
Online

(Speed 4)	✘ ✘ ✘ ✘ ✘ ✘
(Speed 1)	
(Speed 1)	

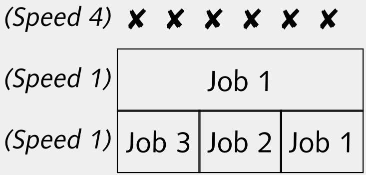
Non-identical machines: it's not that easy

When machines are not identical, migration can be harmful:

Offline



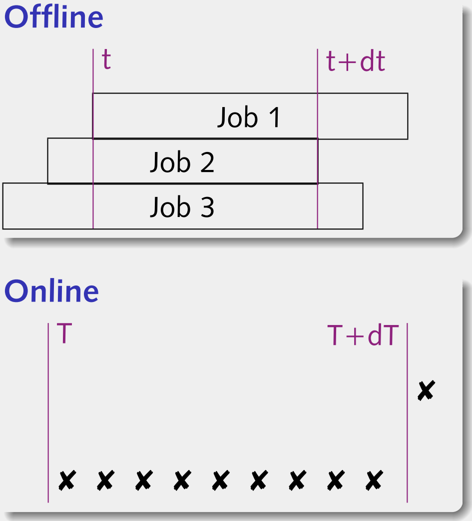
Online



A natural extension to non-identical machines

- Next event offline: $t + dt$.
- Next event online: $T + dT$.
- Schedule "as much as possible" of $[t, t + dt]$ into $[T, T + dT]$.

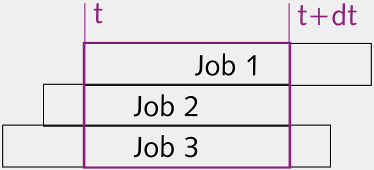
$\Longrightarrow$ at least one interval will be used up



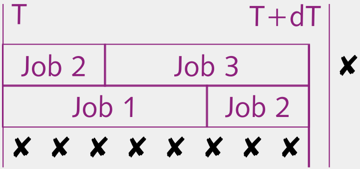
How much is “as much as possible”?

- consider all jobs running in $[t, t + dt]$
- approximately solve $R|pmtn|C_{\max}$
- on online machineset $M(T)$
- fits: ok, exhausts $[t, t + dt]$

Offline



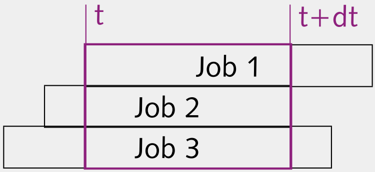
Online



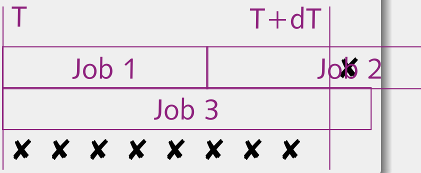
How much is "as much as possible"?

- consider all jobs running in $[t, t + dt]$
- approximately solve $R|pmtn|C_{\max}$
- on online machineset $M(T)$
- doesn't fit: scale solution

Offline



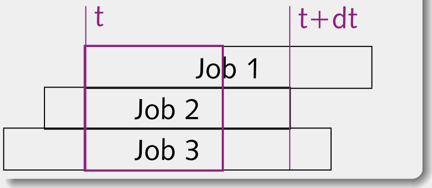
Online



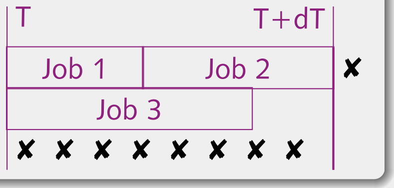
How much is “as much as possible”?

- consider all jobs running in $[t, t+dt]$
- approximately solve $R|pmtn|C_{\max}$
- on online machineset $M(T)$
- doesn't fit: scale solution
- exhaust $[T, T+dT]$

Offline



Online



A different look at related machines

The algorithm *look ahead*

- for the current interval $[t, t + \delta)$:
- find the smallest r such that:
  - Some longest jobs are executed by δ
  - some jobs are shortened to remaining execution time r
  - jobs shorter than r are not executed
  - total area is $m(t)\delta$
- schedule with McNaughton's wraparound rule

Theorem (Albers, Schmidt 99)

look ahead minimizes the makespan for identical machines.

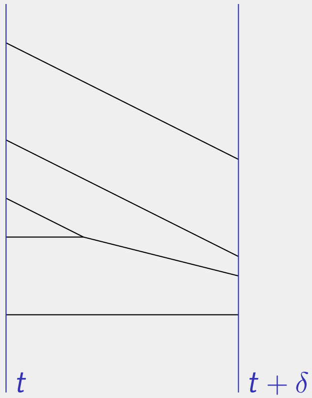
The algorithm, in a picture

timing diagram

- x axis: online time
- y axis: remaining processing time per job
- slope: machine speed

look ahead

- give machines to jobs with highest remaining time
- when lines intersect: jobs “share” the machines for rest of interval



The algorithm, extended to related machines

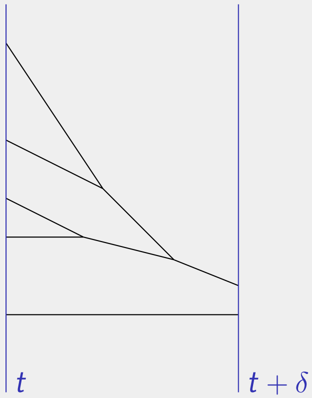
look ahead/fastest machine

- give *fastest* machines to jobs with highest remaining time
- when lines intersect: jobs "share" the machines for rest of interval

By similar proof as before:

Theorem

la/fm minimizes the makespan for related machines.



Summary

- MIMIC: use offline algorithms for online scheduling
- \+ handles release times and precedences
- \+ good bounds for simple stochastic models
- \+ all completion time objectives
- – hard to give bounds for complex stochastic models (aging machines, different reliability)
- – hard to give competitive ratio
- – cannot handle flow time objectives