Online Vector Scheduling

Debmalya Panigrahi
Duke University

Work done with:

Sungjin Im
(UC Merced)

Nat Kell
(Duke)

Janardhan Kulkarni
(MSR → UMN)

Maryam Shadloo
(UC Merced)
Online Load Balancing

Job: 1 2 3 4 5 6 7 8 ...
Load: 1.1 1.3 1.2 0.8 2 0.9 1.2 1.2

(processing time)
Online Load Balancing

[Online Vector Scheduling](#)

**Job:**
- Machine 1
- Machine 2
- Machine m

**Load:**
- Machine 1: 1.1
- Machine 2: 1.2
- Machine m: 1.2

**[Graham ’66]**

- Job 2: 0.8
- Job 3: 1.2
- Job 4: 2
- Job 5: 0.9
- Job 6: 1.2
- Job 7: 1.2
- Job 8: 1.2

(processing time)
Online Load Balancing

[Graham ’66]

Job: 1 2 3
Load:

[Machine 1]
- 1.1 + 1.2 = 2.3
- (load of a machine is the sum of its job loads)

[Machine 2]
- 1.3

[Machine m]

[Machine 4 5 6 7 8 ...]
- 0.8 2 0.9 1.2 1.2
- (processing time)

Online Vector Scheduling
Online Load Balancing

1.1 + 1.2 = 2.3

(load of a machine is the sum of its job loads)

Online problem: cannot see future jobs.

(processing time)
Online Load Balancing

Objective: minimize the makespan of the schedule (maximum load)

Algorithm performance benchmark: Competitive ratio

Online Makespan $\leq \alpha \cdot$ Optimal Makespan $
\implies \alpha$-competitive
Online Load Balancing

**Objectives:** minimize the \( p \)-norm of the machine loads (makespan is the \( \infty \)-norm)

[CW ’75, CC ’76, AAGKKV ’95, AAS ’01, C ’08, CFKKM ’11]

**Machine models:**
- Identical machines (load = \( p_j \))
  [G ’66, FKT ’89, BKR ’94, BFKV ’95, KPT ’96, A ’99, FW ’00, GRTW ’00, R ’01, AAS ’01]
- Related machines (load = \( p_j / s_i \))
  [AAFPW ’97, BCK ’00]
- Unrelated machines (load = \( p_{ij} \))
  [CW ’75, CC ’76, AAGKKV ’95, AAFPW ’97, C ’08, ANR ’95, CFKKM ’11]

### Online Vector Scheduling

**Job:**
1 2 3 4 5 6 7 8

**Load:**
- Machine 1
- Machine 2
- Machine m
How do we load balance simultaneously on multiple resources (e.g., in data centers)?
Online Vector Scheduling

Jobs:

Machine 1

Machine 2

Machine m

Dimension 1 (processor)

Dimension 2 (storage)

Dimension 3 (network)

1

2

3

4

(2, 2.8, 1.3)

(2, 1.5, 1)

(1, 1.5, 1.3)

(1, .8, .9)

...
Online Vector Scheduling

Jobs:

1. (2, 1.5, 1)
2. (1, 1.5, 1.3)
3. (1, .8, .9)
4. ...

Machine 1
Machine 2
Machine m

Dimension 1 (processor)
Dimension 2 (storage)
Dimension 3 (network)
Online Vector Scheduling

Jobs:

Machine 1

Machine 2

Machine m

Dimension 1 (processor)  Dimension 2 (storage)  Dimension 3 (network)

2 + 1 = 3  2.8 + 1.5 = 4.3  1.3 + .9 = 2.2

(loads accumulate in each dimension)

(1, .8, .9)
Online Vector Scheduling

Jobs:

- makespan: maximum over makespan in individual dimensions

Machine 1

Machine 2

Machine m

Dimension 1

Dimension 2

Dimension 3
Online Vector Scheduling

Jobs: $p$-norms: maximum over $p$-norms in individual dimensions
## Summary of Results

<table>
<thead>
<tr>
<th></th>
<th>Makespan minimization</th>
<th>p-norm minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Identical machines</strong></td>
<td>O(log d) [Azar et al ’13, Meyerson et al ’14] Our result: ( \Theta(\log d / \log \log d) )</td>
<td>Our result: ( \Theta((\log d / \log \log d)^{1-1/p}) )</td>
</tr>
<tr>
<td><strong>Unrelated machines</strong> (machine dependent loads)</td>
<td>O(log d + log m) [Meyerson et al ’14] Our result: ( \Theta(\log d + \log m) )</td>
<td>Our result: ( \Theta(\log d + p) )</td>
</tr>
<tr>
<td><strong>Related machines</strong> (non-uniform machine speeds)</td>
<td>Later...</td>
<td>Later...</td>
</tr>
</tbody>
</table>

(Im-Kulkarni-Kell-P. FOCS ’15) (Im-Kell-P.-Shadloo ’17)
### Summary of Results

<table>
<thead>
<tr>
<th></th>
<th>Makespan minimization</th>
<th>(p)-norm minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Identical machines</strong></td>
<td>(O(\log d)) [Azar et al ’13, Meyerson et al ’14]</td>
<td>(\Theta((\log d/\log \log d)^{1-1/p}))</td>
</tr>
<tr>
<td><strong>Our result</strong></td>
<td>(\Theta(\log d/\log \log d))</td>
<td></td>
</tr>
<tr>
<td><strong>Unrelated machines</strong> (machine dependent loads)</td>
<td>(O(\log d + \log m)) [Meyerson et al ’14]</td>
<td>(\Theta(\log d + \log m))</td>
</tr>
<tr>
<td><strong>Our result</strong></td>
<td>(\Theta(\log d + \log m))</td>
<td></td>
</tr>
<tr>
<td><strong>Related machines</strong> (non-uniform machine speeds)</td>
<td>Later...</td>
<td>Later...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Im-Kulkarni-Kell-P. FOCS ’15)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Im-Kell-P.-Shadloo ’17)</td>
</tr>
</tbody>
</table>
Identical machines algorithm: First attempt

Greedy assignment
(minimize maximum load across all machines and dimensions)

unbalanced loads on dimensions...
...can be as bad as poly(d)-competitive
Identical machines algorithm:
First attempt

Random Assignment
(assignment uniformly at random)

Chernoff bounds:
$O(\log(dm))$-competitive
(optimal for unrelated machines)

Greedy assignment
(minimize maximum load across all machines and dimensions)

unbalanced loads on dimensions
…can be as bad as $\text{poly}(d)$-competitive
Algorithm: Random and Greedy

Assign uniformly at random
Algorithm: Random and Greedy

Assign uniformly at random

Online Vector Scheduling
Algorithm: Random and Greedy

Assign uniformly at random
Algorithm: Random and Greedy

Exceeds threshold

Assign uniformly at random

log d/ log log d

log d/ log log d

log d/ log log d

Online Vector Scheduling
Algorithm: Random and Greedy

Assign uniformly at random

Greedy schedule
(minimize max over all machines and dimensions)
Algorithm: Random and Greedy

Assign uniformly at random

Greedy schedule
(minimize max over all machines and dimensions)

E[greedy volume] < volume/poly(d) (Chernoff bounds)

E[greedy makespan] = O(1)
Algorithm: Random and Greedy

Best we can do?
Turns out yes:

Competitive ratio: $O(\log d / \log \log d)$

Ω($\log d / \log \log d$) lower bound
For vector scheduling

Online Vector Scheduling
Online Monochromatic Clique

Given fixed of t colors: red, blue, and green. (here t = 3)

Clique: [Diagram showing a clique]

Objective: minimize the largest monochromatic clique.

i-th vertex arrives: online algorithm gets adjacencies with vertices 1, ..., i-1
Online Monochromatic Clique

Given \textit{fixed} of \( t \) colors: \textit{red}, \textit{blue}, and \textit{green}. (here \( t = 3 \))

\( i \text{th} \) vertex arrives: online algorithm gets adjacencies with vertices \( 1, \ldots, i-1 \)

Objective: minimize the largest monochromatic clique.
Online Monochromatic Clique

Given fixed of $t$ colors: red, blue, and green. (here $t = 3$)

Clique: 

Objective: minimize the largest monochromatic clique.

$i$th vertex arrives: online algorithm gets adjacencies with vertices $1, \ldots, i-1$
Online Monochromatic Clique

Given *fixed* of $t$ colors: red, blue, and green. (here $t = 3$)

Objective: minimize the largest monochromatic clique.

$i$th vertex arrives: online algorithm gets adjacencies with vertices 1, ..., $i-1$
Online Monochromatic Clique

Given *fixed* of $t$ colors: red, blue, and green. (here $t = 3$)

Objective: minimize the largest monochromatic clique.

$i$th vertex arrives: online algorithm gets adjacencies with vertices $1, \ldots, i-1$
Online Monochromatic Clique

Given fixed of t colors: red, blue, and green. (here t = 3)

Objective: minimize the largest monochromatic clique.

ith vertex arrives: online algorithm gets adjacencies with vertices 1, ..., i-1
Online Monochromatic Clique

Given *fixed* of $t$ colors: red, blue, and green. (here $t = 3$)

Objective: minimize the largest monochromatic clique.

$i$th vertex arrives: online algorithm gets adjacencies with vertices $1, ..., i-1$
Online Monochromatic Clique

Given fixed of $t$ colors: red, blue, and green. (here $t = 3$)

Objective: minimize the largest monochromatic clique.

$i$th vertex arrives: online algorithm gets adjacencies with vertices $1, \ldots, i-1$
The Game: Bins versus Colors
(...or robots versus blue devils)

Number of colors: \( t = 4 \)

### Adversary (us)
1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO’s color).
3. Adversary colors the vertex (OPT’s decision)

### Online Algorithm

- My turn!

Bins = algorithm’s coloring.
The Game: Bins versus Colors
(...or robots versus blue devils)

Number of colors: $t = 4$

1. **Adversary defines adjacencies with prior vertices**
2. Algorithm places vertex in a bin (ALGO’s color).
3. Adversary colors the vertex (OPT’s decision)

Bins = algorithm’s coloring
The Game: Bins versus Colors
(...or robots versus blue devils)

Number of colors: \( t = 4 \)

Adversary (us)

1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO’s color).
3. Adversary colors the vertex (OPT’s decision)

Bins = algorithm’s coloring
The Game: Bins versus Colors
(...or robots versus blue devils)

Number of colors: $t = 4$

Adversary (us)

1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO’s color).
3. **Adversary colors the vertex (OPT’s decision)**

Bins = algorithm’s coloring

Online Algorithm

My turn!

![Diagram of Adversary and Online Algorithm with vertices and bins]
The Adversary Strategy

• Split every bin into $\sqrt{t}$ slots: each slot is associated with a distinct set of $\sqrt{t}$ colors
The Construction

1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO’s color).
3. Adversary colors the vertex (OPT’s decision).

vertex \( i \)
The Adversary Strategy

• Split every bin into $\sqrt{t}$ slots: each slot is associated with a distinct set of $\sqrt{t}$ colors

• Generate a “code”: a sequence of strings of length $t$ from a $\sqrt{t}$ alphabet

• For the $i^{th}$ vertex, define adjacencies as follows (say $t = 16$):
  – Suppose the $i^{th}$ string in the code is 1312121121413134
  – Then, add edges to all vertices in slot 1 of bin 1, slot 3 of bin 2, slot 1 of bin 3, etc
The Construction

1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO’s color).
3. Adversary colors the vertex (OPT’s decision).

Code string: 1312121121413134
The Adversary Strategy

• Split every bin into $\sqrt{t}$ slots: each slot is associated with a distinct set of $\sqrt{t}$ colors
• Generate a “code”: a sequence of strings of length $t$ from a $\sqrt{t}$ alphabet
• For the $i^{th}$ vertex, define adjacencies as follows (say $t = 16$):
  – Suppose the $i^{th}$ string in the code is $1312121121413134$
  – Then, add edges to all vertices in slot 1 of bin 1, slot 3 of bin 2, slot 1 of bin 3, etc
  – If the algorithm places the vertex in bin 2, then place it in slot 3 of bin 2
The Construction

1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO’s color).
3. Adversary colors the vertex (OPT’s decision).

Code string: 1312121121413134
The Adversary Strategy

- Split every bin into $\sqrt{t}$ slots: each slot is associated with a distinct set of $\sqrt{t}$ colors
- Generate a “code”: a sequence of strings of length $t$ from a $\sqrt{t}$ alphabet
- For the $i^{th}$ vertex, define adjacencies as follows (say $t = 16$):
  - Suppose the $i^{th}$ string in the code is $1312121121413134$
  - Then, add edges to all vertices in slot 1 of bin 1, slot 3 of bin 2, slot 1 of bin 3, etc
  - If the algorithm places the vertex in bin 2, then place it in slot 3 of bin 2
  - OPT colors the vertex with a color from the $\sqrt{t}$ colors associated with slot 3 that is currently unused in bin 2
The Construction

1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO’s color).
3. Adversary colors the vertex (OPT’s decision).

Code string: 1312121121413134

Bin 15
Color black
The Adversary Strategy

- Split every bin into $\sqrt{t}$ slots: each slot is associated with a distinct set of $\sqrt{t}$ colors.
- Generate a “code”: a sequence of strings of length $t$ from a $\sqrt{t}$ alphabet.
- For the $i^{th}$ vertex, define adjacencies as follows (say $t = 16$):
  - Suppose the $i^{th}$ string in the code is 13121211214131314.
  - Then, add edges to all vertices in slot 1 of bin 1, slot 3 of bin 2, slot 1 of bin 3, etc.
  - If the algorithm places the vertex in bin 2, then place it in slot 3 of bin 2.
  - OPT colors the vertex with a color from the $\sqrt{t}$ colors associated with slot 3 that is currently unused in bin 2.
- Terminate when some slot in some bin has $\sqrt{t}$ vertices.
1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO’s color).
3. Adversary colors the vertex (OPT’s decision).

Observation 1: Algorithm has created a monochromatic $\sqrt{t}$-clique

Observation 2: OPT has perfectly colored the graph in a bin

Lemma (via the probabilistic method): There exist codes that produce only constant-sized monochromatic cliques in OPT
The Construction

1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO’s color).
3. Adversary colors the vertex (OPT’s decision).

\[ \Omega(\sqrt{t}) \text{ lower bound} \]

(t = number colors)
Now for the reduction...

Coloring lower bound

$\Omega(\sqrt{t})$ lower bound

implies

$\Omega(\log d / \log \log d)$ lower bound

For vector scheduling
Using MC Lower Bound for Vector Scheduling

$m = 9$ machines
Issue $m^2 = 81$ jobs

Job dimension $d = \left( \frac{m^2}{\sqrt{m}} \right) = \binom{81}{3}$

Colors correspond to machines
# colors $t = m$
jobs $\leftrightarrow$ vertices

Dimensions correspond to VM size subsets of $\{1, \ldots, m^2\}$
Using MC Lower Bound for Vector Scheduling

\( m = 9 \) machines

Issue \( m^2 = 81 \) jobs

Job dimension \( d = \left( \frac{m^2}{\sqrt{m}} \right) = \binom{81}{3} \)

Colors correspond to machines

# colors \( t = m \)

jobs <-> vertices

Dimensions correspond to Vm size subsets of \{1, ..., m^2\}

(Algorithm's colors)

Colors correspond to machines

(issued by MC instance)
Using MC Lower Bound for Vector Scheduling

\[ m = 9 \text{ machines} \]
Issue \( m^2 = 81 \) jobs

Job dimension \( d = \left( \frac{m^2}{\sqrt{m}} \right) = \left( \frac{81}{3} \right) \)

Colors correspond to machines

# colors \( t = m \) jobs <-> vertices

Dimensions correspond to Vm size subsets of \( \{1, \ldots, m^2\} \)

Online Vector Scheduling
Using MC Lower Bound for Vector Scheduling

\[ m = 9 \text{ machines} \]

Issue \( m^2 = 81 \) jobs

Job dimension \( d = \left( \frac{m^2}{\sqrt{m}} \right) = \binom{81}{3} \)

Colors correspond to machines

Dimensions correspond to Vm size subsets of \( \{1, \ldots, m^2\} \)

\# colors \( t = m \) jobs <-> vertices

(Algorithm’s colors)

(issued by MC instance)

Online Vector Scheduling

\( \{1, 2, 3\} \quad \{1, 2, 4\} \quad \{1, 2, 5\} \quad \{2, 3, 6\} \quad \{2, 4, 6\} \quad \{79, 80, 81\} \)

1 iff vertex forms a clique with previous vertices in the set
Using MC Lower Bound for Vector Scheduling

\[ m = 9 \text{ machines} \]
Issue \( m^2 = 81 \) jobs

Job dimension \( d = \left( \frac{m^2}{\sqrt{m}} \right) = \left( \frac{81}{3} \right) \)

Colors correspond to machines
Dimensions correspond to \( \sqrt{m} \) size subsets of \{1, \ldots, m^2\}

Colors correspond to machines

(Algorithm’s colors)

# colors \( t = m \)
jobs <-> vertices

1 iff vertex forms a clique with previous vertices in the set

(Algorithm’s colors)

Colors correspond to machines
Dimensions correspond to \( \sqrt{m} \) size subsets of \{1, \ldots, m^2\}

Online Vector Scheduling
Using MC Lower Bound for Vector Scheduling

\[ m = 9 \] machines

Issue \( m^2 = 81 \) jobs

Job dimension \( d = \left( \frac{m^2}{\sqrt{m}} \right) = \binom{81}{3} \)

Colors correspond to machines

Dimensions correspond to VM size subsets of \( \{1, \ldots, m^2\} \)

# colors \( t = m \)

jobs <-> vertices

(Algorithm’s colors)

(issued by MC instance)

1 iff vertex forms a clique with previous vertices in the set

Online Vector Scheduling

\( \{1, 2, 3\} \) \( \{1, 2, 4\} \) \( \{1, 2, 5\} \) \( \{2, 3, 6\} \) \( \{2, 4, 6\} \) \( \{79, 80, 81\} \)
Using MC Lower Bound for Vector Scheduling

\[ m = 9 \text{ machines} \]

Issue \( m^2 = 81 \) jobs

Job dimension \( d = \left( \frac{m^2}{\sqrt{m}} \right) = \left( \frac{81}{3} \right) \)

1. After \( m^2 \) vertices, there will exist a monochromatic clique of size \( \sqrt{m} \) on some color \( c \).
2. \( \Rightarrow \) dimension corresponding to these vertices will have a load of \( \sqrt{m} \) on machine \( c \).
3. Size of largest monochromatic clique in OPT’s graph coloring is \( O(1) \).
4. ALGO/OPT \( \Rightarrow \Omega(\sqrt{m}) = \Omega(\log d / \log \log d) \)
## Summary of Results

<table>
<thead>
<tr>
<th></th>
<th>Makespan minimization</th>
<th>p-norm minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Identical machines</strong></td>
<td>$O(\log d)$</td>
<td>$\Theta(\frac{\log d}{\log \log d})$</td>
</tr>
<tr>
<td></td>
<td>[Azar <em>et al</em> ’13, Meyerson <em>et al</em> ’14]</td>
<td>Our result: $\Theta(\frac{\log d}{\log \log d})$</td>
</tr>
<tr>
<td></td>
<td>Our result: $\Theta(\frac{\log d}{\log \log d})$</td>
<td>Our result: $\Theta((\log d/\log \log d)^{1-1/p})$</td>
</tr>
<tr>
<td><strong>Unrelated machines</strong></td>
<td>$O(\log d + \log m)$</td>
<td>$\Theta(\log d + p)$</td>
</tr>
<tr>
<td>(machine dependent loads)</td>
<td>[Meyerson <em>et al</em> ’14]</td>
<td>(Im-Kulkarni-Kell-P. FOCS ’15)</td>
</tr>
<tr>
<td></td>
<td>Our result: $\Theta(\log d + \log m)$</td>
<td>Our result: $\Theta(\log d + p)$</td>
</tr>
<tr>
<td><strong>Related machines</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(non-uniform machine speeds)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Im-Kell-P.-Shadloo ‘17)</td>
</tr>
</tbody>
</table>
Related Machines (homogenous)

Processing time = load/speed

Jobs:

1

(2, 2.8, 1.3)

2

(2, 1.5, 1)

...
Related Machines (homogenous)

Execution time = load/speed

Jobs:

Machine 1
speed = 1

2

Machine m
speed = 1/2

2.8

(2, 1.5, 1)

Online Vector Scheduling
Related Machines (homogenous)

Execution time = load/speed

Jobs:

Machine 1
speed = 1

Machine m
speed = 1/2

Online Vector Scheduling
Related Machines (heterogeneous)

Processing time = load/speed

Jobs:

1
(2, 2.8, 1.3)

2
(2, 1.5, 1)

...
Related Machines (heterogeneous)

Processing time = load/speed

Jobs:

- Machine 1
  - speed = 1
  - Processing time = load/speed
  - (2, 1.5, 1)

- Machine m
  - speed = 1/2
  - speed = 1/3
  - speed = 1/4

Online Vector Scheduling
Related Machines (heterogeneous)

Processing time = load/speed

Jobs:

Machine 1
- speed = 1
- Load = 2

Machine m
- speed = 1/2
- Load = 4

...
## Summary of Results

<table>
<thead>
<tr>
<th></th>
<th>Makespan minimization</th>
<th>p-norm minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Identical machines</strong></td>
<td>(O(\log d))</td>
<td>Our result: (\Theta(\log d/\log \log d))</td>
</tr>
<tr>
<td></td>
<td>([\text{Azar et al '13, Meyerson et al '14}])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{Our result: } \Theta(\log d/\log \log d))</td>
<td></td>
</tr>
<tr>
<td><strong>Unrelated machines</strong></td>
<td>(O(\log d + \log m))</td>
<td>Our result: (\Theta(\log d + p))</td>
</tr>
<tr>
<td>(machine dependent loads)</td>
<td>([\text{Meyerson et al '14}])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{Our result: } \Theta(\log d + \log m))</td>
<td></td>
</tr>
<tr>
<td><strong>Related machines</strong></td>
<td>(\Theta(\log d/\log \log d))</td>
<td>Our result: (O(\log^3 d))</td>
</tr>
<tr>
<td>(non-uniform machine speeds)</td>
<td></td>
<td>(Im-Kell-P.-Shadloo ’17)</td>
</tr>
<tr>
<td></td>
<td>(\text{Our result: } \Theta(\log d + \log m))</td>
<td></td>
</tr>
<tr>
<td><strong>Heterogeneous</strong></td>
<td>(\text{})</td>
<td>(\text{})</td>
</tr>
</tbody>
</table>

Online Vector Scheduling
# Summary of Results

<table>
<thead>
<tr>
<th></th>
<th>Makespan minimization</th>
<th>p-norm minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Identical machines</strong></td>
<td>O(log d)</td>
<td>Our result: Θ((log d/log log d) ^ (1 - 1/p))</td>
</tr>
<tr>
<td></td>
<td>[Azar et al '13, Meyerson et al '14]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Our result: Θ(log d/log log d)</td>
<td></td>
</tr>
<tr>
<td><strong>Unrelated machines</strong></td>
<td>O(log d + log m)</td>
<td>Our result: Θ(log d + p)</td>
</tr>
<tr>
<td>(machine dependent loads)</td>
<td>[Meyerson et al '14]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Our result: Θ(log d + log m)</td>
<td></td>
</tr>
<tr>
<td><strong>Related machines</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(non-uniform machine speeds)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Homogeneous</strong></td>
<td>Θ(log d/log log d)</td>
<td>O(log^3 d)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Im-Kulkarni-Kell-P. FOCS '15)</td>
</tr>
<tr>
<td><strong>Heterogeneous</strong></td>
<td>Θ(log d + log m)</td>
<td>Θ(log d + p)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Im-Kell-P.-Shadloo '17)</td>
</tr>
</tbody>
</table>

First O(1) competitive for d = 1
Machine Grouping

Want to reduce problem to identical machines...
Natural to try to group machines of similar speed.

**Issue:** if total speed (processing power) of faster machines is large, slower machines go unutilized.
Machine Smoothing

- speed = 1
- speed = 2/3
- speed = 1/2
- speed = 2/5
- speed = 2/5
- speed = 2/5
- speed = 2/5
- speed = 2/5
- speed = 1/3
- speed = 1/3
- speed = 1/3

Machine Smoothing

Group 0: (total speed 1)

- Group machines so that total speed increases exponentially.
- Replace machines with identical machines with (roughly) same total speed.
Machine Smoothing

Group 0:
- speed = 1
- speed = 2/3
- speed = 1/2
- speed = 2/5

(total speed 1)

Group 1:
- speed = 1/2
- speed = 1/2
- speed = 1/2
- speed = 1/2

(total speed 2)

• Group machines so that total speed increases exponentially.
• Replace machines with identical machines with (roughly) same total speed.
Machine Smoothing

- Group machines so that total speed increases exponentially.
- Replace machines with identical machines with (roughly) same total speed.

Group 0:  
- speed = 1
- speed = 2/3
- speed = 1/2
- speed = 1/3

Group 1:  
- speed = 1/2
- speed = 1/2
- speed = 1/2
- speed = 1/3

Group 2:  
- speed = 2/5
- speed = 2/5
- speed = 1/3
- speed = 1/3

(total speed 1)
(total speed 2)
(total speed 4)

Online Vector Scheduling
Machine Smoothing

- Group machines so that total speed increases exponentially.
- Replace machines with identical machines with (roughly) same total speed.

Lemma (informal): Any schedule on a smoothed instance can be replicated on the original instance with constant change in makespan, and vice-versa. A similar statement can be shown for all p-norms as well.
Makespan minimization:
Slowest fit on Smoothed Instance

Suppose OPT = 10

Jobs:
1
(3, 3, 1)

2
(2, 1.5, 1)

... 

Algorithm: Assign to slowest group such that all execution times are \( \leq c \cdot \text{OPT} \)
Makespan minimization: Slowest fit on Smoothed Instance

Suppose OPT = 10

Jobs:

Algorithm: Assign to slowest group such that all execution times are <= c. OPT

Online Vector Scheduling
Makespan minimization: 
Slowest fit on Smoothed Instance

Suppose $OPT = 10$

Algorithm: Assign to slowest group such that all execution times are $\leq c \cdot OPT$

Jobs:

$speed = 1$

$speed = 1/2$

$speed = 1/4$

$(2, 1.5, 1)$

$(3, 3, 1)$
Makespan minimization: Slowest fit on Smoothed Instance

Suppose $\text{OPT} = 10$

Jobs:

Algorithm: Assign to slowest group such that all processing times are $\leq c \cdot \text{OPT}$

.... Then, assign jobs using the identical machines algorithm (within each group).
p-norm minimization

Challenge: Even if we are able to guess OPT, how do we divide it among the machine groups?

Indeed, no algorithm previously known even for $d = 1$
p-norm minimization

Challenge: Even if we are able to guess OPT, how do we divide it among the machine groups?

Indeed, no algorithm previously known even for $d = 1$

Algorithm has two interleaved stages:
- fractional solution via gradient descent on a potential defined by a suitable fractional relaxation
- online rounding uses a slowest-fit strategy on the fractional solution
Thank You

Questions?