Scheduling algorithms
for heterogeneous and failure-prone platforms

Yves Robert
Ecole Normale Supérieure de Lyon
& Institut Universitaire de France

http://graal.ens-lyon.fr/~yrobert

RENPAR, Saint-Malo, 11 Mai 2011
Introduction  Who needs a scheduler?  Background: scheduling DAGs  Steady-state scheduling  Energy-aware scheduling  Scheduling with replication  Checkpointing strategies  Conclusion

Ce qui a changé en 25 ans?
Ce qui a changé en 25 ans? **Nous . . .**
Ce qui a changé en 25 ans? Nous ... L'humour ...
Nous... L’humour... Et les machines parallèles!

From (good old) parallel architectures...
Nous ... L’humour ... Et les machines parallèles!

... to heterogeneous clusters ...
Nous ... L’humour ... Et les machines parallèles!

... to large-scale grid platforms ...
Nous . . . L’humour . . . Et les machines parallèles!

... everything hard-to-program ...  

*(courtesy Rajeev Thakur)*
Nous ... L’humour ... Et les machines parallèles!

... and definitely prone to failure!  

(courtesy Al Geist)

---

Fear of the Exponential Growth in Parallelism

- Fundamental assumptions of today’s system software architecture did not anticipate exponential growth in parallelism.
- Number of system components is increasing faster than component reliability, which is set by COTS needs.
- Number of components and MTBF leading to a paradigm shift – Faults will be the norm rather than rare events. SW adaptation to frequent failures or “be crazy and die like a dog”
New platforms, new problems, new solutions

Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines.

On heterogeneous multicore failure-prone platforms, it gets worse 😞

Need to adapt algorithms and scheduling strategies: new objective functions, new models.
New platforms, new problems, **new solutions**

Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines.

**On heterogeneous multicore failure-prone platforms, it gets worse 😞**

Need to adapt algorithms and scheduling strategies: new objective functions, new models.
Outline

1. Who needs a scheduler?
2. Background: scheduling DAGs
3. Steady-state scheduling
4. Energy-aware scheduling
5. Scheduling with replication
6. Checkpointing strategies
7. Conclusion
Outline

1. Who needs a scheduler?
2. Background: scheduling DAGs
3. Steady-state scheduling
4. Energy-aware scheduling
5. Scheduling with replication
6. Checkpointing strategies
7. Conclusion
Who needs a scheduler?

- Billions of (mostly idle) computers in the world
- All interconnected by these (mostly empty) network pipes
- Resources: abundant and cheap, not to say unlimited and free

- Virtual machines
- Greedy resource selection
- Demand-driven execution: first-come first-serve, round-robin

Nobody needs a scheduler!!!
Who needs a scheduler?

- Billions of (mostly idle) computers in the world
- All interconnected by these (mostly empty) network pipes
- Resources: abundant and cheap, not to say unlimited and free
- Virtual machines
- Greedy resource selection
- Demand-driven execution: first-come first-serve, round-robin

Let's prove this wrong?!
(A) Independent tasks – no communication

- $B$ independent equal-size tasks
- $p$ processors $P_1, P_2, \ldots, P_p$
- $w_i =$ time for $P_i$ to process a task

**Intuition:** load of $P_i$ proportional to its speed $1/w_i$

- Assign $n_i$ tasks to $P_i$

**Objective:** minimize $T_{\text{exe}} = \max_{\sum_{i=1}^p n_i=B} (n_i \times w_i)$
Dynamic programming

**With 3 processors:** \( w_1 = 3, \ w_2 = 5, \text{ and } w_3 = 8 \)

<table>
<thead>
<tr>
<th>Task</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
<th>( T_{exe} )</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>
Dynamic programming

*With 3 processors: $w_1 = 3$, $w_2 = 5$, and $w_3 = 8*

<table>
<thead>
<tr>
<th>Task</th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
<th>$T_{exe}$</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

**With 3 processors:** \( w_1 = 3, \ w_2 = 5, \text{ and } w_3 = 8 \)

<table>
<thead>
<tr>
<th>Task</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
<th>( T_{\text{exe}} )</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

With 3 processors: \( w_1 = 3, w_2 = 5, \text{ and } w_3 = 8 \)

<table>
<thead>
<tr>
<th>Task</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
<th>( T_{\text{exe}} )</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

With 3 processors: \( w_1 = 3, w_2 = 5, \text{ and } w_3 = 8 \)

<table>
<thead>
<tr>
<th>Task</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
<th>( T_{\text{exe}} )</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

With 3 processors: $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$

<table>
<thead>
<tr>
<th>Task</th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
<th>$T_{exe}$</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

**With 3 processors:** \( w_1 = 3, \ w_2 = 5, \text{ and } w_3 = 8 \)

<table>
<thead>
<tr>
<th>Task</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
<th>( T_{\text{exe}} )</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

With 3 processors: \( w_1 = 3, \ w_2 = 5, \text{ and } w_3 = 8 \)

<table>
<thead>
<tr>
<th>Task</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
<th>( T_{\text{exe}} )</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

**With 3 processors:** $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$

<table>
<thead>
<tr>
<th>Task</th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
<th>$T_{exe}$</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

With 3 processors: $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$

<table>
<thead>
<tr>
<th>Task</th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
<th>$T_{exe}$</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

With 3 processors: $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$

<table>
<thead>
<tr>
<th>Task</th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
<th>$T_{\text{exe}}$</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Static versus dynamic

- Greedy (demand-driven) would have done a perfect job
- Would even be better (possible variations in processor speeds)

Static assignment required useless thinking 😞
Static versus dynamic

- Greedy (demand-driven) would have done a perfect job
- Would even be better (possible variations in processor speeds)

Static assignment required **useless thinking 😞**
(B) With dependencies – still no communication

**A simple finite difference problem**

- Iteration space: 2D rectangle of size $N_1 \times N_2$
- Dependences between tiles $\left\{ \left( \begin{array}{c} 1 \\ 0 \end{array} \right), \left( \begin{array}{c} 0 \\ 1 \end{array} \right) \right\}$
Allocation strategy (1/3)

Use column-wise allocation to enhance locality

Stepwise execution
Allocation strategy (1/3)

Use column-wise allocation to enhance locality

\[
\begin{align*}
\ldots & \\
6 & \rightarrow 6 \\
5 & \rightarrow 6 \\
4 & \rightarrow 5 & \rightarrow 6 \\
3 & \rightarrow 4 & \rightarrow 5 \\
2 & \rightarrow 3 & \rightarrow 4 \\
1 & \rightarrow 2 & \rightarrow 3 & \ldots
\end{align*}
\]

Stepwise execution
Allocation strategy (1/3)

Use column-wise allocation to enhance locality

Stepwise execution
Allocation strategy (1/3)

Use column-wise allocation to enhance locality

Stepwise execution
Allocation strategy (1/3)

Use column-wise allocation to enhance locality

Stepwise execution
Allocation strategy (1/3)

Use column-wise allocation to enhance locality

Stepwise execution

Yves.Robert@ens-lyon.fr
Allocation strategy (1/3)

Use column-wise allocation to enhance locality

... 6 5 → 6 4 → 5 → 6 3 → 4 → 5 2 → 3 → 4 1 → 2 → 3 ...
With column-wise allocation,

\[ T_{\text{opt}} \approx \frac{N_1 \times N_2}{\sum_{i=1}^{P} \frac{1}{w_i}}. \]

- Greedy (demand-driven) allocation ⇒ slowdown ?!
- Execution progresses at the pace of the slowest processor 😞
Allocation strategy (2/3)

- With column-wise allocation,
  \[ T_{\text{opt}} \approx N_1 \times N_2 \sum_{i=1}^{p} \frac{1}{w_i} \]

- Greedy (demand-driven) allocation ⇒ slowdown ?!

- Execution progresses at the pace of the slowest processor 😞
Allocation strategy (3/3)

With 3 processors, $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$:

$$\begin{align*}
T_{\text{exe}} & \approx \frac{8}{3} N_1 N_2 \approx 2.67 N_1 N_2 \\
T_{\text{opt}} & \approx \frac{120}{79} N_1 N_2 \approx 1.52 N_1 N_2
\end{align*}$$
Periodic static allocation (1/2)

With 3 processors, $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$:

Assigning blocks of $B = 10$ columns, $T_{\text{exe}} \approx 1.6 \ N_1 N_2$
Periodic static allocation (2/2)

- $L = \text{lcm}(w_1, w_2, \ldots, w_p)$
  
  Example: $L = \text{lcm}(3, 5, 8) = 120$

- $P_1$ receives first $n_1 = L/w_1$ columns, $P_2$ next $n_2 = L/w_2$ columns, and so on

- Period: block of $B = n_1 + n_2 + \ldots + n_p$ contiguous columns
  
  Example: $B = n_1 + n_2 + n_3 = 40 + 24 + 15 = 79$

- **Change schedule:**
  - Sort processors so that $n_1 w_1 \leq n_2 w_2 \leq \ldots \leq n_p w_p$
  - Process horizontally within blocks

- **Optimal 😊**
Outline

1. Who needs a scheduler?
2. Background: scheduling DAGs
3. Steady-state scheduling
4. Energy-aware scheduling
5. Scheduling with replication
6. Checkpointing strategies
7. Conclusion
Traditional scheduling – Framework

- **Application** DAG $G = (\mathcal{T}, \mathcal{E}, w)$
  - $\mathcal{T} = \text{set of tasks}$
  - $\mathcal{E} = \text{dependence constraints}$
  - $w(T) = \text{computational cost of task } T \text{ (execution time)}$
  - $c(T, T') = \text{communication cost (data sent from } T \text{ to } T')$

- **Platform** Set of $p$ identical processors

- **Schedule**
  - $\sigma(T) = \text{date to begin execution of task } T$
  - alloc($T$) = processor assigned to it

- **Objective** **Makespan** or total execution time

$$MS(\sigma) = \max_{T \in \mathcal{T}} (\sigma(T) + w(T))$$
Traditional scheduling – Constraints

- **Data dependences** If \((T, T') \in E\) then
  
  - if \(alloc(T) = alloc(T')\) then \(\sigma(T) + w(T) \leq \sigma(T')\)
  
  - if \(alloc(T) \neq alloc(T')\) then \(\sigma(T) + w(T) + c(T, T') \leq \sigma(T')\)

- **Resource constraints**

\[
alloc(T) = alloc(T') \Rightarrow \begin{align*}
(\sigma(T) + w(T) &\leq \sigma(T')) \text{ or } (\sigma(T') + w(T') \leq \sigma(T))
\end{align*}
\]
Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- **Very crude** for network resources
  - Unlimited number of simultaneous sends/receives per processor
  - No contention → unbounded bandwidth on any link
  - Fully connected interconnection graph (clique)
- In fact, model assumes *infinite* network capacity
**Makespan minimization**

- **NP-hardness**
  - $Pb(p)$ NP-complete for independent tasks and no communications
    - $(E = \emptyset, p = 2$ and $c = 0)$
  - $Pb(p)$ NP-complete for UET-UCT graphs $(w = c = 1)$

- **Approximation algorithms**
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no $\lambda$-approximation in general
List scheduling – Without communications

Initialization:

- Compute priority level of all tasks
- Priority queue = list of free tasks (tasks without predecessors) sorted by priority

While there remain tasks to execute:

- Add new free tasks, if any, to the queue.
- If there are \( q \) available processors and \( r \) tasks in the queue, remove first \( \min(q, r) \) tasks from the queue and execute them

Priority level

- Use critical path: longest path from the task to an exit node
- Computed recursively by a bottom-up traversal of the graph
List scheduling – Without communications

Initialization:
- Compute priority level of all tasks
- Priority queue = list of free tasks (tasks without predecessors) sorted by priority

While there remain tasks to execute:
- Add new free tasks, if any, to the queue.
- If there are $q$ available processors and $r$ tasks in the queue, remove first $\min(q, r)$ tasks from the queue and execute them

Priority level
- Use critical path: longest path from the task to an exit node
- Computed recursively by a bottom-up traversal of the graph
List scheduling – With communications

- Priority level
  - Use **pessimistic** critical path: include all edge costs in the weight
  - Computed recursively by a bottom-up traversal of the graph

- MCP **Modified Critical Path**
  - Assign free task with highest priority to **best** processor
  - Best processor = finishes execution first, given already taken scheduling decisions
  - Free tasks may not be ready for execution (communication delays)
  - May explore inserting the task in empty slots of schedule
  - Complexity $O(|V| \log |V| + (|E| + |V|)p)$
Extending the model to heterogeneous resources

- Task graph with $n$ tasks $T_1, \ldots, T_n$.
- Platform with $p$ heterogeneous processors $P_1, \ldots, P_p$.
- Computation costs:
  - $w_{iq} =$ execution time of $T_i$ on $P_q$
  - $\overline{w_i} = \frac{\sum_{q=1}^{p} w_{iq}}{p}$ average execution time of $T_i$
  - particular case: consistent tasks $w_{iq} = w_i \times \gamma_q$
- Communication costs:
  - $\text{data}(i,j)$: data volume for edge $e_{ij} : T_i \rightarrow T_j$
  - $\nu_{qr}$: communication time for unit-size message from $P_q$ to $P_r$ (zero if $q = r$)
  - $\text{com}(i,j,q,r) = \text{data}(i,j) \times \nu_{qr}$ communication time from $T_i$ executed on $P_q$ to $T_j$ executed on $P_r$
  - $\overline{\text{com}_{ij}} = \text{data}(i,j) \times \frac{\sum_{1 \leq q,r \leq p, q \neq r} \nu_{qr}}{p(p-1)}$ average communication cost for edge $e_{ij} : T_i \rightarrow T_j$
HEFT: Heterogeneous Earliest Finish Time

- **Priority level:**
  - \( \text{rank}(T_i) = \bar{w}_i + \max_{T_j \in \text{Succ}(T_i)} (\text{com}_{ij} + \text{rank}(T_j)) \),
  - where \( \text{Succ}(T) \) is the set of successors of \( T \)
  - Recursive computation by bottom-up traversal of the graph

- **Allocation**
  - For current task \( T_i \), determine best processor \( P_q \):
    - minimize \( \sigma(T_i) + w_{iq} \)
  - Enforce constraints related to communication costs
  - Insertion scheduling: look for \( t = \sigma(T_i) \) s.t. \( P_q \) is available during interval \([t, t + w_{iq}]\)

- **Complexity:** same as MCP without/with insertion
What’s wrong?

- 😊 Nothing (still may need to map a DAG onto a platform!)
- 😞 Absurd communication model:
  complicated: many parameters to instantiate
  while not realistic (clique + no contention)
- 😞 Wrong metric: need to relax makespan minimization objective
What’s wrong?

- 😊 Nothing (still may need to map a DAG onto a platform!)
- 😞 Absurd communication model: complicated: many parameters to instantiate while not realistic (clique + no contention)
- 😞 Wrong metric: need to relax makespan minimization objective
What’s wrong?

- 😊 Nothing (still may need to map a DAG onto a platform!)
- 😞 Absurd communication model:
  complicated: many parameters to instantiate
  while not realistic (clique + no contention)
- 😞 Wrong metric: need to relax makespan minimization objective
Favorite communication models

**One-port**
- serialize incoming (and/or) outgoing communications
- pessimistic but realistic
- standard MPI

**Bounded multi-port**
- several concurrent incoming and outgoing communications
- share total available bandwidth
- bottleneck = network card
- multi-threaded systems
Outline

1. Who needs a scheduler?
2. Background: scheduling DAGs
3. Steady-state scheduling
4. Energy-aware scheduling
5. Scheduling with replication
6. Checkpointing strategies
7. Conclusion
Volunteer computing applications

- Monte Carlo methods
- SETI@home
- Factoring large numbers
- Searching for Mersenne primes
- Particle detection at CERN (LHC@home)
- ... many others: BOINC at http://boinc.berkeley.edu
Example
Example

A is the root of the tree; all tasks start at A

Time for computing one task in C

Time for sending one task from A to B

Yves.Robert@ens-lyon.fr
Example
Example

A compute
A send

B receive
B compute

C receive
C compute
C send

D receive
D compute

Time→
Example

A compute
A send
B receive
B compute
C receive
C compute
C send
D receive
D compute

Time→

1 2 3
Example

A compute
A send
B receive
B compute
C receive
C compute
C send
D receive
D compute

Time→

1 2 3

Yves.Robert@ens-lyon.fr
Introduction

Who needs a scheduler?

Background: scheduling DAGs

Steady-state scheduling

Energy-aware scheduling

Scheduling with replication

Checkpointing strategies

Conclusion

Example

![Diagram of a computation graph with nodes and edges representing tasks and their execution over time. The diagram shows the sequence of compute, send, receive, and compute operations for nodes A, B, C, and D, with each operation labeled as A compute, A send, B receive, B compute, C receive, C compute, C send, D receive, and D compute. The time axis is marked from 1 to 3.](image-url)
Example

Steady-state: 7 tasks every 6 time units

Steady-state scheduling

Yves.Robert@ens-lyon.fr
Rule of the game

- Master sends tasks **sequentially**, without preemption
- Full computation/communication overlap for each worker
- Worker $P_i$ receives a task in $c_i$ time-units
- Worker $P_i$ processes a task in $w_i$ time-units
Worker $P_i$ executes $\alpha_i$ tasks per time-unit

Computations: $\alpha_i w_i \leq 1$

One-port communications: $\sum_i \alpha_i c_i \leq 1$

Objective: maximize throughput

$$\rho = \sum_i \alpha_i$$
Solution

- Faster-communicating workers first: \( c_1 \leq c_2 \leq \ldots \)
- Make full use of first \( q \) workers, where \( q \) largest index s.t. 
  \[
  \sum_{i=1}^{q} \frac{c_i}{w_i} \leq 1
  \]
- Make partial use of next worker \( P_{q+1} \)
- **Discard** other workers

**Bandwidth-centric strategy**
- Delegate work to the fastest communicating workers
- It doesn’t matter if these workers are computing slowly
- Slow workers will not contribute much to overall throughput
Example

\textbf{Tasks}\n\begin{align*}
6 \text{ tasks to } P_1 & \quad 6c_1 = 6 \\
3 \text{ tasks to } P_2 & \quad 3c_2 = 6 \\
2 \text{ tasks to } P_3 & \quad 2c_3 = 6
\end{align*}

\textbf{Communication}\n\begin{align*}
6w_1 & = 18 \\
3w_2 & = 18 \\
2w_3 & = 2
\end{align*}

\textbf{Computation}\n\begin{align*}
11 \text{ tasks every 18 time-units } (\rho = 11/18 \approx 0.6)
\end{align*}
Example

Fully active

Discarded

🎉 Compare to purely greedy (demand-driven) strategy!

5 tasks every 36 time-units ($\rho = 5/36 \approx 0.14$)

Even if resources are cheap and abundant, resource selection is key to performance
Extension to trees

Resource selection based on local information (children)
Does this really work?

- Can we deal with arbitrary platforms (including cycles)?
- Can we deal with return messages?
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?
Does this really work?

- Can we deal with arbitrary platforms (including cycles)? **Yes**
- Can we deal with return messages?
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?
Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages?
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?
Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages? Yes
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?
Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages? Yes
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?
Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages? Yes
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)? Yes, I mean, almost!
LP formulation still works well ...

Conservation law

∀m, n \[ \sum_j \text{sent}(P_j \rightarrow P_i, e_{mn}) + \text{executed}(P_i, T_m) \]
\[ = \text{executed}(P_i, T_n) + \sum_k \text{sent}(P_i \rightarrow P_k, e_{mn}) \]

Computations

\[ \sum_m \text{executed}(P_i, T_m) \times \text{flops}(T_m) \times w_i \leq 1 \]

Outgoing communications

\[ \sum_{m,n} \sum_j \text{sent}(P_j \rightarrow P_i, e_{mn}) \times \text{bytes}(e_{mn}) \times c_{ij} \leq 1 \]
but schedule reconstruction is harder

- ☑ Actual (cyclic) schedule obtained in polynomial time
- ☑ Asymptotic optimality
- ☹ A couple of practical problems (large period, # buffers)
- ☹ No local scheduling policy
The beauty of steady-state scheduling

**Rationale**  Maximize throughput

**Simplicity**  Relaxation of makespan minimization problem

- Ignore initialization and clean-up
- Precise ordering of tasks/messages not needed
- Characterize resource activity per time-unit:
  - fraction of time spent computing for each application
  - fraction of time spent receiving from / sending to each neighbor?

**Efficiency**  Optimal throughput $\Rightarrow$ optimal schedule (up to a constant number of tasks)

Periodic schedule, described in compact form

$\Rightarrow$ compiling a loop instead of a DAG!
Lesson learnt?

Resource selection is mandatory
Outline

1. Who needs a scheduler?
2. Background: scheduling DAGs
3. Steady-state scheduling
4. Energy-aware scheduling
5. Scheduling with replication
6. Checkpointing strategies
7. Conclusion
Scheduling workflows

- **Data streams**: images, frames, matrices, etc.
- **Structured applications**: several steps to process each data set

Diagram:
- Original images
- Pre-processing
- Encoding
- Final result

Goal: “efficiently” use computing resources
Example

- 4 processing stages, 3 processors at our disposal
- Where/how can we execute the application?
Example

- 4 processing stages, 3 processors at our disposal
- Where/how can we execute the application?
Example

- 4 processing stages, 3 processors at our disposal
- Where/how can we execute the application?

Use all resources greedily
Many communications to pay, not efficient at all!
Example

- 4 processing stages, 3 processors at our disposal
- Where/how can we execute the application?

Everything on fastest processor: no communication
Optimal execution time to process one single data
4 processing stages, 3 processors at our disposal

Where/how can we execute the application?

Optimal throughput: processing different data in parallel

Resource selection: do not use slowest processor
Optimization criteria

- **Period** $P$: time interval between the beginning of execution of two consecutive data sets (inverse of throughput)
- **Latency** $L$: maximal time elapsed between beginning and end of execution of a data set
- **Reliability**: inverse of $F$, probability of failure of the application (i.e., some data sets will not be processed)
- **Energy** $E$: total power dissipated by platform
Optimization criteria

- **Period** $P$: time interval between the beginning of execution of two consecutive data sets (inverse of throughput)
- **Latency** $L$: maximal time elapsed between beginning and end of execution of a data set
- **Reliability**: inverse of $F$, probability of failure of the application (i.e., some data sets will not be processed)
- **Energy** $E$: total power dissipated by platform
Optimization criteria

- **Period** $\mathcal{P}$: time interval between the beginning of execution of two consecutive data sets (inverse of throughput)
- **Latency** $\mathcal{L}$: maximal time elapsed between beginning and end of execution of a data set
- **Reliability**: inverse of $\mathcal{F}$, probability of failure of the application (i.e., some data sets will not be processed)
- **Energy** $\mathcal{E}$: total power dissipated by platform
Optimization criteria

- **Period** $P$: time interval between the beginning of execution of two consecutive data sets (inverse of throughput)
- **Latency** $L$: maximal time elapsed between beginning and end of execution of a data set
- **Reliability**: inverse of $F$, probability of failure of the application (i.e., some data sets will not be processed)
- **Energy** $E$: total power dissipated by platform

“The internet begins with coal”
Algorithmic techniques:

- Shut down idle processors
- **Dynamic speed scaling**: processors can run at variable speed, e.g., Intel XScale, Intel Speed Step, AMD PowerNow
- **VDD Hopping**: from discrete modes to continuous speed

Dissipated power

- The higher the speed, the higher the power consumption
- $Power = f \times V^2$, and $V$ (voltage) increases with $f$ (frequency)
- Speed $s$: $P(s) = s^\alpha + P_{static}$, with $2 \leq \alpha \leq 3$
A primer on energy consumption

Algorithmic techniques:
- Shut down idle processors
- **Dynamic speed scaling**: processors can run at variable speed, e.g., Intel XScale, Intel Speed Step, AMD PowerNow
- **VDD Hopping**: from discrete modes to continuous speed

Dissipated power
- The higher the speed, the higher the power consumption
- $Power = f \times V^2$, and $V$ (voltage) increases with $f$ (frequency)
- Speed $s$: $P(s) = s^\alpha + P_{static}$, with $2 \leq \alpha \leq 3$
Example

Period: $T = 3$
Latency: $L = 8$
Example

\[ T = 3 \quad L = 8 \]

- **Period:** \( T = 3 \)
- **Latency:** \( L = 8 \)
Example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Example

\[ P = 3^3 + 8^3 = 539 \]

**Period:** \( T = 3 \)

**Latency:** \( L = 8 \)
Example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Introduction

Who needs a scheduler?

Background: scheduling DAGs

Steady-state scheduling

Energy-aware scheduling

Scheduling with replication

Checkpointing strategies

Conclusion

Example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Example

\[ P = 3^3 + 8^3 = 539 \]

Period: \( T = 3 \)
Latency: \( L = 8 \)
Example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Example

\[ P = 3^3 + 8^3 = 539 \]

Period: \( T = 3 \)
Latency: \( L = 8 \)
Example

Period: $T = 3$
Latency: $L = 8$

$P = 3^3 + 8^3 = 539$
Example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Example

Period: \( T = 3 \)
Latency: \( L = 8 \)

\[ P = 3^3 + 8^3 = 539 \]
Example

- Period: $T = 3$
- Latency: $L = 8$

$$P = 3^3 + 8^3 = 539$$
Example

\[ P = 3^3 + 8^3 = 539 \]

Period: \( T = 3 \)
Latency: \( L = 8 \)
Example

- **Period:** $T = 3$
- **Latency:** $L = 8$

$P = 539$

$P = 8$
Example

- Period: $T = 3$, $T = 15$
- Latency: $L = 8$
Example

- Period: $P_1 = 3$ \hspace{1cm} $T_1 = 15$
- Latency: $L_1 = 8$ \hspace{1cm} $L_1 = 17$

$P = 539$

$P = 8$
Multi-criteria optimization

Performance-oriented objectives

- Minimize period (inverse of throughput)
- Minimize latency (time to process a data set)
- Minimize application failure probability

Environmental objectives

- Minimize energy consumption
- Minimize platform cost

Objective function

- Minimize $\alpha P + \beta L + \gamma E$
- Minimize period, given a response time bound and an energy budget!
Multi-criteria optimization

Performance-oriented objectives

- Minimize period (inverse of throughput) $P$
- Minimize latency (time to process a data set) $L$
- Minimize application failure probability

Environmental objectives

- Minimize energy consumption $E$
- Minimize platform cost

Objective function

- Minimize $\alpha P + \beta L + \gamma E$?
Multi-criteria optimization

Performance-oriented objectives

- Minimize period (inverse of throughput) $\mathcal{P}$
- Minimize latency (time to process a data set) $\mathcal{L}$
- Minimize application failure probability

Environmental objectives

- Minimize energy consumption $\mathcal{E}$
- Minimize platform cost

Objective function

- Minimize $\alpha \cdot \mathcal{P} + \beta \cdot \mathcal{L} + \gamma \cdot \mathcal{E}$
- Minimize period, given a response time bound and an energy budget!
Research problems

Single workflow

- Performance (period/latency) vs. environment (energy, cost)
- Robust mappings (replication for performance vs. for reliability)

Several (concurrent) workflows

- Competition for CPU and network resources
- Fairness between applications:
  - max-min throughput
  - min-max stretch (slowdown factor)
- Sensitivity to application/platform parameter changes
Research problems

Single workflow

- Performance (period/latency) vs. environment (energy, cost)
- Robust mappings (replication for performance vs. for reliability)

Several (concurrent) workflows

- Competition for CPU and network resources
- Fairness between applications:
  - max-min throughput
  - min-max stretch (slowdown factor)
- Sensitivity to application/platform parameter changes
Outline

1. Who needs a scheduler?
2. Background: scheduling DAGs
3. Steady-state scheduling
4. Energy-aware scheduling
5. Scheduling with replication
6. Checkpointing strategies
7. Conclusion
Cycle-stealing scenario (1/2)

- Execute 4 jobs A, B, C, D during week-end
- Replicate them on 3 machines $P_1$, $P_2$ and $P_3$
- Risk increases with time
- Machines reclaimed at 8am on Monday with probability 1
Cycle-stealing scenario (1/2)

- Execute 4 jobs A, B, C, D during week-end
- Replicate them on 3 machines $P_1$, $P_2$ and $P_3$
- Risk increases **linearly** with time
- Machines reclaimed at 8am on Monday with probability 1
Cycle-stealing scenario (2/2)

\[ P_1 \quad 1 \quad 2 \quad 3 \quad 4 \]

A  B  C  D
Cycle-stealing scenario (2/2)

A B C D

\[ P_1 \quad 1 \quad 2 \quad 3 \quad 4 \]

\[ P_2 \]
## Cycle-stealing scenario (2/2)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$P_2$</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Cycle-stealing scenario (2/2)
Cycle-stealing scenario (2/2)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$P_2$</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
### Cycle-stealing scenario (2/2)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$P_2$</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$P_2$</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Problem

- $n$ chunks (divisible computational workload)
- $p$ identical computers
- Replicate $p$ times the execution of each chunk

Unrecoverable (fail-stop) interruptions
A-priori knowledge of risk (failure probability)

**Goal:** maximize expected amount of work done
Problem

- \( n \) chunks (divisible computational workload)
- \( p \) identical computers
- Replicate \( p \) times the execution of each chunk

Unrecoverable (fail-stop) interruptions
- A-priori knowledge of risk (failure probability)

Goal: maximize expected amount of work done
Symmetric schedules (1/2)

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Symmetric schedules (1/2)

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
## Symmetric schedules (1/2)

### Table

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P_1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>(P_2)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(P_3)</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>(P_4)</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

### Diagram

- Group 1
- Group 1
- Group 3
Symmetric schedules (1/2)

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Symmetric schedules (1/2)

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Symmetric schedules (1/2)

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Group 1

Group 1

Group 3
Symmetric schedules (1/2)

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Symmetric schedules (1/2)

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
\hline
\text{Group 1} & \text{Group 1} & \text{Group 3}
\end{array}
\]

\[
\begin{array}{cccccccccccc}
\text{Time} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
P_1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 4 & 4 & 4 \\
P_2 & 2 & 2 & 2 & 3 & 3 & 3 & 4 & 4 & 4 & 1 & 1 & 1 \\
P_3 & 3 & 3 & 3 & 4 & 4 & 4 & 1 & 1 & 1 & 2 & 2 & 2 \\
P_4 & 4 & 4 & 4 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3
\end{array}
\]
Symmetric schedules (1/2)

\[
\begin{array}{cccc}
\text{Group 1} & \text{Group 1} & \text{Group 3} \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
\text{Time} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
P_1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 4 & 4 & 4 \\
P_2 & 2 & 2 & 2 & 3 & 3 & 3 & 4 & 4 & 4 & 1 & 1 & 1 \\
P_3 & 3 & 3 & 3 & 4 & 4 & 4 & 1 & 1 & 1 & 2 & 2 & 2 \\
P_4 & 4 & 4 & 4 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \\
\end{array}
\]
Symmetric schedules (2/2)

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Group 1 | Group 2 | Group 3
---|---|---
1 | 2 | 3
6 | 5 | 4
9 | 8 | 7
12 | 11 | 10

Time-steps for group execution

$G_{ij} = i$-th execution of group $j$
# Optimization problem (1/2)

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>
Optimization problem (1/2)

Group 1      Group 2      Group 3
  1          2          3
  6          5          4
  9          8          7
 12         11         10

All four executions fail with probability prop. to $1 \times 6 \times 9 \times 12$
Optimization problem (1/2)

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

All four executions fail with probability prop. to $1 \times 6 \times 9 \times 12$

- First execution failure probability $\lambda \omega G_{11}$
  
- Second execution failure probability $\lambda \omega G_{21}$
  
- Third execution failure probability $\lambda \omega G_{31}$
  
- Fourth execution failure probability $\lambda \omega G_{41}$

Whole schedule failure probability $\lambda^4 \omega^4 \prod_{i=1}^{4} G_{i1}$

$G_{11} = 1$

$G_{21} = 6$

$G_{31} = 9$

$G_{41} = 12$

$1 \times 6 \times 9 \times 12$
Optimization problem (1/2)

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

All four executions fail with probability prop. to $2 \times 5 \times 8 \times 11$
Optimization problem (1/2)

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

All four executions fail with probability prop. to $3 \times 4 \times 7 \times 10$
Optimization problem (2/2)

\[ \mathbb{E}(\text{jobdone}) = \sum_{j=1}^{\frac{n}{p}} p \omega \left( 1 - \lambda^p \omega^p \prod_{i=1}^{p} G_{i,j} \right) \]

**Problem**

Minimize \( K = \sum_{j=1}^{\frac{n}{p}} \prod_{i=1}^{p} G_{i,j} \)

where entries of \( G \) are a permutation of \([1..n]\)

**Bound**

\[ K_{\text{min}} = \left\lfloor \frac{n}{p} \cdot \frac{n^{p/\sqrt{n}}}{\sqrt{n!}} \right\rfloor \]
Heuristics (1/3)

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

(a) Cyclic: $K = 3104$

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

(b) Reverse: $K = 2368$
### Heuristics (2/3)

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

**(c) Mirror:** $K = 2572$

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

**(d) Snake:** $K = 2464$
Heuristics (3/3)

<table>
<thead>
<tr>
<th>Step 1</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCP</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 2</th>
<th>6</th>
<th>5</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCP</td>
<td>6</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 3</th>
<th>9</th>
<th>8</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCP</td>
<td>54</td>
<td>80</td>
<td>84</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 4</th>
<th>12</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCP</td>
<td>54</td>
<td>80</td>
<td>84</td>
</tr>
</tbody>
</table>

(e) Greedy: $K = 2368$

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

(f) Optimal $K = 2352 \geq K_{min} = \lceil 3^{3/2}\sqrt{12!} \rceil = 2348$
### A nice little algorithmic challenge

Fill up matrix with a permutation of $[1..n]$ minimizing the sum of column products

<table>
<thead>
<tr>
<th></th>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
<th>...</th>
<th>Group $n/p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>...</td>
<td>x</td>
</tr>
<tr>
<td>$P_2$</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>...</td>
<td>x</td>
</tr>
<tr>
<td>$P_3$</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>...</td>
<td>x</td>
</tr>
<tr>
<td>$P_4$</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>...</td>
<td>x</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$P_p$</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>...</td>
<td>x</td>
</tr>
</tbody>
</table>

OK, OK, Greedy asymptotically optimal 😊
Still, this is a 😞 frustrating 😞 open 😞 problem 😞 😞 😞
# A nice little algorithmic challenge

<table>
<thead>
<tr>
<th></th>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
<th>...</th>
<th>Group ( n/p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>...</td>
<td>x</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>...</td>
<td>x</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>...</td>
<td>x</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>( P_p )</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>...</td>
<td>x</td>
</tr>
</tbody>
</table>

Fill up matrix with a **permutation of \([1..n]\)** minimizing the **sum of column products**

OK, OK, Greedy asymptotically optimal 😊
Still, this is a 😞 frustrating 😞 open 😞 problem 😞 😞 😞
Many potential sources of heterogeneity

- Processor of different speeds
- Links of different bandwidths
- Resources obeying different risk functions
  - different owner categories?

Trade-off between speed and reliability
Mapping applications on volatile resources

Iterative applications
- Independent / tightly-coupled tasks within each iteration
- Synchronization (checkpoint) after each iteration

Master-worker paradigm
- Heterogeneous processors
- Limited available bandwidth from master to workers

Volatile desktop grids
- Resource availability: $\text{UP} / \text{RECLAIMED} / \text{DOWN}$
- Markov process (with different transition probabilities)

Goal: on-line policies for resource selection
- Which resources to enroll? fast or reliable?
- When to change configuration?
Outline

1. Who needs a scheduler?
2. Background: scheduling DAGs
3. Steady-state scheduling
4. Energy-aware scheduling
5. Scheduling with replication
6. Checkpointing strategies
7. Conclusion
Motivation

Framework

- **Very very** large number of processing elements (e.g., $2^{20}$)
- Failure-prone platform (like any realistic platform)
- Large application to be executed

$\implies$ Failure(s) will indeed occur before completion!

Questions

- When to checkpoint the application?
- Always use all processors?
Notations

- Overall size of work: $W$
- Checkpoint cost: $C$
- Downtime: $D$
  - software rejuvenation via rebooting
  - replacement by spare
- Recovery cost after failure: $R$
Makespan vs. NextFailure

- **Makespan**: Minimize job’s expected makespan
- **NextFailure**: Maximize expected amount of work completed before next failure

**Rationale:**
- **NextFailure**: optimization on a “failure-by-failure” basis
- Hopefully a good approximation, at least for large job sizes $\mathcal{W}$
\begin{align*}
T(0|\tau) &= 0 \\
T(\omega|\tau) &= \begin{cases} \\
\omega_1 + C + T(\omega - \omega_1|\tau + \omega_1 + C) \\
T_{\text{wasted}}(\omega_1 + C|\tau) + T(\omega|R) \\
\text{if the processor does not fail during} \\
\text{the next } \omega_1 + C \text{ units of time} \\
\text{otherwise.}
\end{cases}
\end{align*}

\textbf{Makespan:} find } \omega_1 = f(\omega|\tau) \leq \omega \text{ that minimizes } \mathbb{E}(T(W|0))
NextFailure

\[ W(0|\tau) = 0 \]
\[ W(\omega|\tau) = \begin{cases} 
  \omega_1 + W(\omega - \omega_1|\tau + \omega_1 + C) \\
  0 
\end{cases} \]
if the processor does not fail during
the next \( \omega_1 + C \) units of time,
otherwise.

**NextFailure**: find \( \omega_1 = f(\omega|\tau) \leq \omega \) that maximizes
\( E(W(W|0)) \)
Single-processor jobs

**Makespan**
- Exponential $F(t) = 1 - e^{-\lambda t}$: periodic strategy optimal
- Other (Weibull $F(t) = 1 - e^{-\frac{t^k}{\lambda^k}}$): open

**NextFailure**
- Open for all failure laws

**Accurate dynamic programming approximations**
Parallel jobs

Models

- Embarrassingly parallel jobs: $\mathcal{W}(p) = \mathcal{W} / p$
- Amdahl parallel jobs: $\mathcal{W}(p) = \mathcal{W} / p + \gamma \mathcal{W}$
- Numerical kernels: $\mathcal{W}(p) = \mathcal{W} / p + \gamma \mathcal{W}^{2/3} / \sqrt{p}$

Accurate dynamic programming approximations
Parallel jobs

\[ \text{loss}(p_{\text{used}}) \text{ vs } p_{\text{used}} \text{ for jobs obeying Amdahl's law on Jaguar} \]
Problems

- Software/hardware techniques to reduce checkpoint, recovery, migration times and to improve failure prediction
- "Self-fault-tolerant" algorithms (e.g. asynchronous iterative)
- Multi-criteria scheduling problem 
  throughput/energy/reliability
- Add replication and group execution

Need combine all these approaches! 😞
Tools for the road

- Forget absolute makespan minimization
- Resource selection mandatory
- Divisible load (fractional tasks)
- Single application: period / latency / energy / robustness
- Several applications: max-min fairness, MAX stretch
- Linear programming: absolute bound to assess heuristics
- Probabilities and stochastic models . . . unavoidable
Scheduling for large-scale platforms

- If platform is well identified and relatively stable, try to:
  (i) accurately model hierarchical structure
  (ii) design efficient/robust/energy-aware scheduling algorithms

- If platform is not stable enough, or if it evolves too fast, dynamic schedulers are the only option

- Otherwise, grab any opportunity to

  inject static knowledge into dynamic schedulers

- Is this opportunity a niche?
- Does it encompass a wide range of applications?
Scheduling for large-scale platforms

- If platform is well identified and relatively stable, try to:
  (i) accurately model hierarchical structure
  (ii) design efficient/robust/energy-aware scheduling algorithms

- If platform is not stable enough, or if it evolves too fast, dynamic schedulers are the only option

- Otherwise, grab any opportunity to inject static knowledge into dynamic schedulers

اتهمشس

🤔 Is this opportunity a niche?
😊 Does it encompass a wide range of applications?
THUS, FOR ANY NONDETERMINISTIC TURING MACHINE M THAT RUNS IN SOME POLYNOMIAL TIME $p(n)$, WE CAN DEVISE AN ALGORITHM THAT TAKES AN INPUT $\omega$ OF LENGTH $n$ AND PRODUCES $E_m,\omega$. THE RUNNING TIME IS $O(p^2(n))$ ON A MULTITAPE DETERMINISTIC TURING MACHINE AND...

WTF, MAN. I JUST WANTED TO LEARN HOW TO PROGRAM VIDEO GAMES.