Overview of Scheduling 2/2

Loris MARCHAL

(with the help of Olivier BEAUMONT, Henri CASANOVA,
Arnaud LEGRAND, Yves ROBERT and Frédéric VIVIEN)

GRAAL project,
Laboratoire de l'Informatique du Parallélisme,
École Normale Supérieure de Lyon, France.

`loris.marchal@ens-lyon.fr`

ACIS Laboratory
September 27, 2006
http://graal.ens-lyon.fr/~lmarchal/talks.html

Outline

1. Background on traditional scheduling
2. Divisible Load Scheduling
3. Steady-State Scheduling
4. Simulation for Grid Computing

Outline

1. Background on traditional scheduling
2. Divisible Load Scheduling
3. Steady-State Scheduling
4. Simulation for Grid Computing

Traditional Scheduling – Summary

- Scheduling graph of tasks on processors
- For regular parallel computers:
  - homogeneous processors
  - infinite network capacity
- Difficult problems (list scheduling heuristics)
- When including heterogeneity: no guaranteed algorithms
- $\leadsto$ model too acurate to be tractable on heterogeneous platforms

Outline

1. Background on traditional scheduling
2. **Divisible Load Scheduling**
3. Steady-State Scheduling
4. Simulation for Grid Computing

Divisible Load Scheduling – Summary

- Changing the task model:
  - graph of tasks $\rightsquigarrow$ one perfectly divisible task
- Considering simple platforms:
  - master-slave, bus or star networks
- Results:
  - Compute optimal makespan
  - Study the impact of processor ordering
  - Point out solution shape
    (all processors enrolled, same termination time)
  - Compute optimal allocation
  - Adapt to tree platforms,. . .
- Limitations:
  - Very simple application model
  - Simple communication scheme
  - Multi-round algorithms not tractable (NP-hard)

Divisible Load Scheduling – Summary

- Changing the task model:
  - graph of tasks $\rightsquigarrow$ one perfectly divisible task
- Considering simple platforms:
  - master-slave, bus or star networks
- Results:
  - Compute optimal makespan
  - Study the impact of processor ordering
  - Point out solution shape
    (all processors enrolled, same termination time)
  - Compute optimal allocation
  - Adapt to tree platforms,. . .
- Limitations:
  - Very simple application model
  - Simple communication scheme
  - Multi-round algorithms not tractable (NP-hard)

Divisible Load Scheduling – Summary

- Changing the task model:
  - graph of tasks $\rightsquigarrow$ one perfectly divisible task
- Considering simple platforms:
  - master-slave, bus or star networks
- Results:
  - Compute optimal makespan
  - Study the impact of processor ordering
  - Point out solution shape
    (all processors enrolled, same termination time)
  - Compute optimal allocation
  - Adapt to tree platforms,. . .
- Limitations:
  - Very simple application model
  - Simple communication scheme
  - Multi-round algorithms not tractable (NP-hard)

Outline

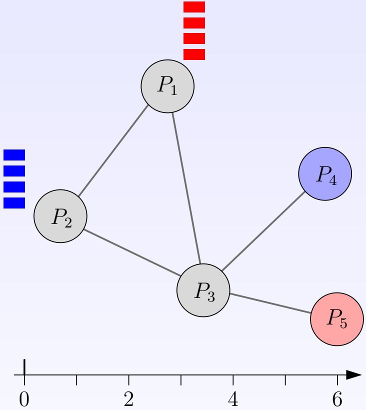
1. Background on traditional scheduling
2. Divisible Load Scheduling
3. Steady-State Scheduling
    - Packet routing
    - Master-slave tasking
4. Simulation for Grid Computing

Steady-State Scheduling

Changing the objective:

- Makespan minimization: reasonable for small set of tasks
- On distributed heterogeneous platforms: large amount of work
- No difference if program runs for 3 hours or 3 hours + 5 secondes
- Total completion time may not be the right metric
- Efficient resource utilization during steady-state:
  throughput maximization
- Neglect initialization and clean-up phases

Packet routing without fixed path

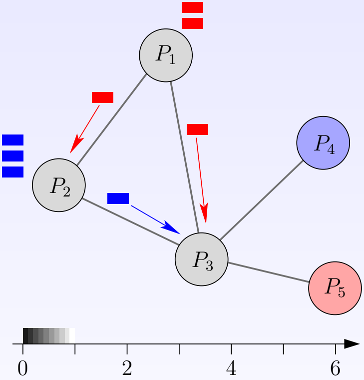


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path

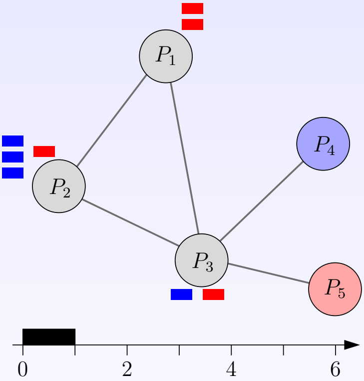


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l) | n^{k,l} > 0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path

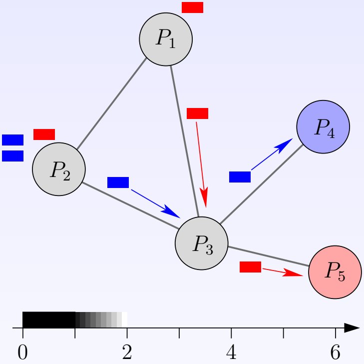


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path

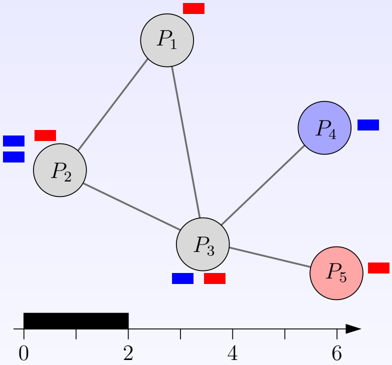


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l) | n^{k,l} > 0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path

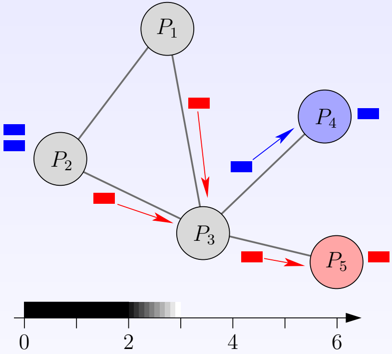


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l) | n^{k,l} > 0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path

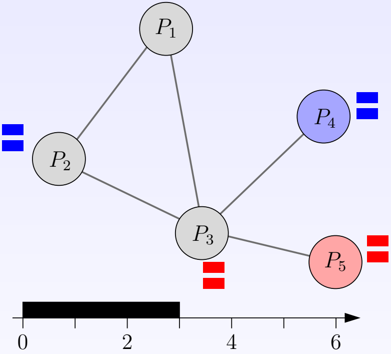


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path

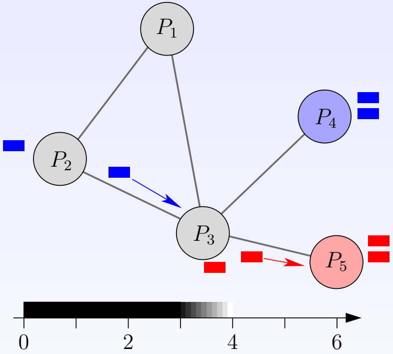


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n^{k,l}_{i,j}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l) | n^{k,l} > 0} n^{k,l}_{i,j} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path

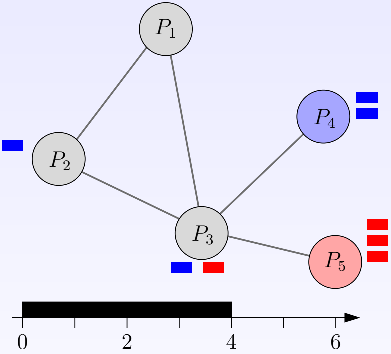


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path

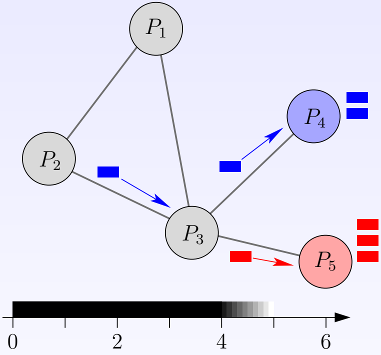


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path

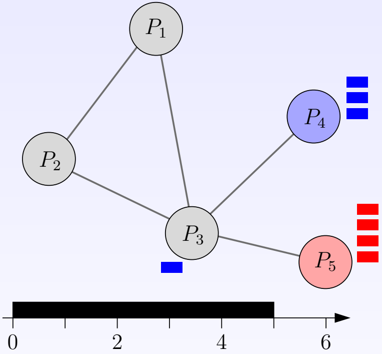


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path

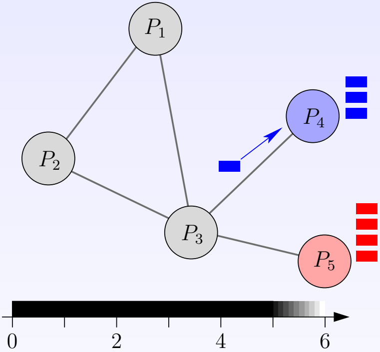


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path

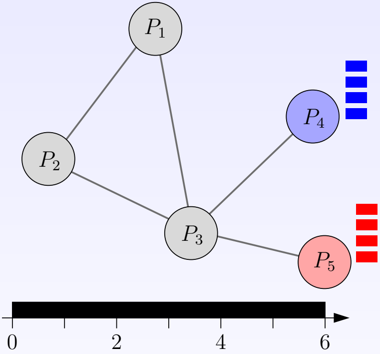


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path

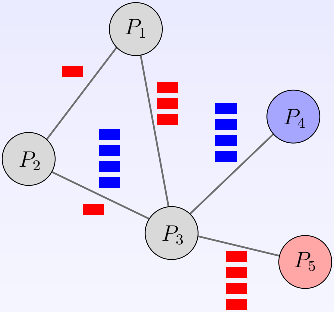


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l) | n^{k,l} > 0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path

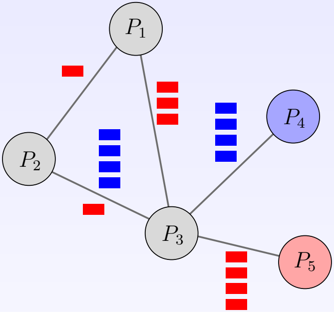


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n^{k,l}_{i,j}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l) | n^{k,l} > 0} n^{k,l}_{i,j} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

Packet routing without fixed path



- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n^{k,l}_{i,j}$: total number of packets routed from k to l and crossing edge (i,j)
- Congestion:

$$C_{i,j} = \sum_{(k,l) | n^{k,l} > 0} n^{k,l}_{i,j} \qquad\qquad C_{\max} = \max_{i,j} C_{i,j}$$

Equations (1/2)

1. Initialization

$$\sum_{j|(k,j)\in A} n_{k,j}^{k,l} = n^{k,l}$$

2. Reception

$$\sum_{i|(i,l)\in A} n_{i,l}^{k,l} = n^{k,l}$$

3. Conservation law

$$\sum_{i|(i,j)\in A} n_{i,j}^{k,l} = \sum_{i|(j,i)\in A} n_{j,i}^{k,l} \quad \forall (k,l), j \neq k, j \neq l$$

Equations (1/2)

1. Initialization

$$\sum_{j|(k,j)\in A} n_{k,j}^{k,l} = n^{k,l}$$

2. Reception

$$\sum_{i|(i,l)\in A} n_{i,l}^{k,l} = n^{k,l}$$

3. Conservation law

$$\sum_{i|(i,j)\in A} n_{i,j}^{k,l} = \sum_{i|(j,i)\in A} n_{j,i}^{k,l} \quad \forall (k,l), j \neq k, j \neq l$$

Equations (1/2)

1. Initialization

$$\sum_{j|(k,j)\in A} n_{k,j}^{k,l} = n^{k,l}$$

2. Reception

$$\sum_{i|(i,l)\in A} n_{i,l}^{k,l} = n^{k,l}$$

3. Conservation law

$$\sum_{i|(i,j)\in A} n_{i,j}^{k,l} = \sum_{i|(j,i)\in A} n_{j,i}^{k,l} \quad \forall (k,l), j \neq k, j \neq l$$

Equations (2/2)

4. Congestion

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}$$

5. Objective function

$$C_{\max} \geq C_{i,j}, \qquad \forall i, j$$

Minimize $C_{\max}$

Linear program in rational numbers: polynomial-time solution. In practice use Maple, Mupad, lp-solve,. . .

Solution:
number of messages $n_{i,j}^{k,l}$ of each edge to minimize total congestion

Equations (2/2)

4. Congestion

$$C_{i,j} = \sum_{(k,l) | n^{k,l} > 0} n_{i,j}^{k,l}$$

5. Objective function

$$C_{\max} \geq C_{i,j}, \qquad \forall i, j$$

Minimize $C_{\max}$

Linear program in rational numbers: polynomial-time solution. In practice use Maple, Mupad, lp-solve,. . .

Solution:
number of messages $n_{i,j}^{k,l}$ of each edge to minimize total congestion

Equations (2/2)

4. Congestion

$$C_{i,j} = \sum_{(k,l) | n^{k,l} > 0} n_{i,j}^{k,l}$$

5. Objective function

$$C_{\max} \geq C_{i,j}, \qquad \forall i, j$$

Minimize $C_{\max}$

Linear program in rational numbers: polynomial-time solution. In practice use Maple, Mupad, lp-solve,...

Solution:
number of messages $n_{i,j}^{k,l}$ of each edge to minimize total congestion

Equations (2/2)

4. Congestion

$$C_{i,j} = \sum_{(k,l) | n^{k,l} > 0} n_{i,j}^{k,l}$$

5. Objective function

$$C_{\max} \geq C_{i,j}, \qquad \forall i, j$$

Minimize $C_{\max}$

Linear program in rational numbers: polynomial-time solution. In practice use Maple, Mupad, lp-solve,...

Solution:
number of messages $n_{i,j}^{k,l}$ of each edge to minimize total congestion

Routing algorithm

1. Computing optimal solution $C_{\max}$ of previous linear program
2. Consider periods of length Ω (to be defined later)
3. During each time-interval $[p\Omega, (p+1)\Omega]$, follow the optimal solution: edge (i, j) forwards:

   $$m_{i,j}^{k,l} = \left\lfloor \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} \right\rfloor \qquad \text{packets that go from } k \text{ to } l. \text{ (if available)}$$

4. number of such periods: $\left\lceil \frac{C_{\max}}{\Omega} \right\rceil$
5. After time-step

   $$T = \left\lceil \frac{C_{\max}}{\Omega} \right\rceil \Omega \leq C_{\max} + \Omega$$

   sequentially process M residual packets in no longer than ML time-steps, where L is the maximum length of a simple path in the network

Routing algorithm

1. Computing optimal solution $C_{\max}$ of previous linear program
2. Consider periods of length Ω (to be defined later)
3. During each time-interval $[p\Omega, (p+1)\Omega]$, follow the optimal solution: edge (i,j) forwards:

$$m_{i,j}^{k,l} = \left\lfloor \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} \right\rfloor$$

packets that go from k to l.
(if available)

4. number of such periods: $\left\lceil \frac{C_{\max}}{\Omega} \right\rceil$
5. After time-step

$$T \equiv \left\lceil \frac{C_{\max}}{\Omega} \right\rceil \Omega \leq C_{\max} + \Omega$$

sequentially process M residual packets in no longer than ML time-steps, where L is the maximum length of a simple path in the network

Routing algorithm

1. Computing optimal solution $C_{\max}$ of previous linear program
2. Consider periods of length Ω (to be defined later)
3. During each time-interval $[p\Omega, (p+1)\Omega]$, follow the optimal solution: edge (i,j) forwards:

$$m_{i,j}^{k,l} = \left\lfloor \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} \right\rfloor$$

packets that go from k to l.
(if available)

4. number of such periods: $\left\lceil \frac{C_{\max}}{\Omega} \right\rceil$
5. After time-step

$$T \equiv \left\lceil \frac{C_{\max}}{\Omega} \right\rceil \Omega \leq C_{\max} + \Omega$$

sequentially process M residual packets in no longer than ML time-steps, where L is the maximum length of a simple path in the network

Routing algorithm

1. Computing optimal solution $C_{\max}$ of previous linear program
2. Consider periods of length Ω (to be defined later)
3. During each time-interval $[p\Omega, (p+1)\Omega]$, follow the optimal solution: edge (i, j) forwards:

$$m_{i,j}^{k,l} = \left\lfloor \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} \right\rfloor$$

packets that go from k to l.
(if available)

4. number of such periods: $\left\lceil \frac{C_{\max}}{\Omega} \right\rceil$
5. After time-step

$$T \equiv \left\lceil \frac{C_{\max}}{\Omega} \right\rceil \Omega \leq C_{\max} + \Omega$$

sequentially process M residual packets in no longer than ML time-steps, where L is the maximum length of a simple path in the network

Routing algorithm

1. Computing optimal solution $C_{\max}$ of previous linear program
2. Consider periods of length Ω (to be defined later)
3. During each time-interval $[p\Omega, (p+1)\Omega]$, follow the optimal solution: edge (i, j) forwards:

$$m_{i,j}^{k,l} = \left\lfloor \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} \right\rfloor$$

packets that go from k to l.
(if available)

4. number of such periods: $\left\lceil \frac{C_{\max}}{\Omega} \right\rceil$
5. After time-step

$$T \equiv \left\lceil \frac{C_{\max}}{\Omega} \right\rceil \Omega \leq C_{\max} + \Omega$$

sequentially process M residual packets in no longer than ML time-steps, where L is the maximum length of a simple path in the network

Feasibility

$$\sum_{(k,l)} m_{i,j}^{k,l} \leq \sum_{(k,l)} \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} = \frac{C_{i,j}\Omega}{C_{\max}} \leq \Omega$$

Makespan

- Define Ω as $\Omega = \sqrt{C_{\max} n_c}$.
- Total number of packets still inside network at time-step T is at most

$$2|A|\sqrt{C_{\max} n_c} + |A| n_c$$

- Makespan:

$$C_{\max} \leq C^* \leq C_{\max} + \sqrt{C_{\max} n_c} + 2|A|\sqrt{C_{\max} n_c}|V| + |A| n_c |V|$$

$$C^* = C_{\max} + O(\sqrt{C_{\max}})$$

Makespan

- Define Ω as $\Omega = \sqrt{C_{\max} n_c}$.
- Total number of packets still inside network at time-step T is at most
$$2|A|\sqrt{C_{\max} n_c} + |A| n_c$$
- Makespan:
$$C_{\max} \leq C^* \leq C_{\max} + \sqrt{C_{\max} n_c} + 2|A|\sqrt{C_{\max} n_c}|V| + |A| n_c |V|$$
$$C^* = C_{\max} + O(\sqrt{C_{\max}})$$

Makespan

- Define Ω as $\Omega = \sqrt{C_{\max} n_c}$.
- Total number of packets still inside network at time-step T is at most

$$2|A|\sqrt{C_{\max} n_c} + |A|n_c$$

- Makespan:

$$C_{\max} \leq C^* \leq C_{\max} + \sqrt{C_{\max} n_c} + 2|A|\sqrt{C_{\max} n_c}|V| + |A|n_c|V|$$

$$C^* = C_{\max} + O(\sqrt{C_{\max}})$$

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not *needed*

[illegible]

- which (rational) fraction of time is spent computing for which application?
- which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period
⇒ react on the fly to resource availability variations

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed
- Characterize resource activity during each time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

⇒ react on the fly to resource availability variations

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed
- Characterize resource activity during each time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

⇒ react on the fly to resource availability variations

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed
- Characterize resource activity during each time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

⇒ react on the fly to resource availability variations

Steady-state scheduling

Background	Approach pioneered by Bertsimas and Gamarnik
Rationale	Maximize throughput (total load executed per period)
Simplicity	Relaxation of makespan minimization problem • Ignore initialization and clean-up phases • Precise ordering/allocation of tasks/messages not needed • Characterize resource activity during each time-unit: - which (rational) fraction of time is spent computing for which application? - which (rational) fraction of time is spent receiving or sending to which neighbor?
Efficiency	Periodic schedule, described in compact form
Adaptability	Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period ⇒ react on the fly to resource availability variations

Steady-state scheduling

Background	Approach pioneered by Bertsimas and Gamarnik
Rationale	Maximize throughput (total load executed per period)
Simplicity	Relaxation of makespan minimization problem • Ignore initialization and clean-up phases • Precise ordering/allocation of tasks/messages not needed • Characterize resource activity during each time-unit: - which (rational) fraction of time is spent computing for which application? - which (rational) fraction of time is spent receiving or sending to which neighbor?
Efficiency	Periodic schedule, described in compact form
Adaptability	Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period ⇒ react on the fly to resource availability variations

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed
- Characterize resource activity during each time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

⇒ react on the fly to resource availability variations

Steady-state scheduling

Background	Approach pioneered by Bertsimas and Gamarnik
Rationale	Maximize throughput (total load executed per period)
Simplicity	Relaxation of makespan minimization problem • Ignore initialization and clean-up phases • Precise ordering/allocation of tasks/messages not needed • Characterize resource activity during each time-unit: - which (rational) fraction of time is spent computing for which application? - which (rational) fraction of time is spent receiving or sending to which neighbor?
Efficiency	Periodic schedule, described in compact form
Adaptability	Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period ⇒ react on the fly to resource availability variations

Master-slave platform

Master-slave tasking Simple yet efficient

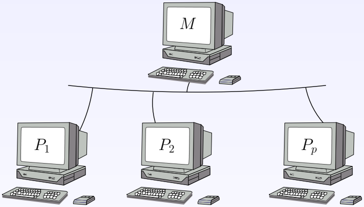
Standard implementation Independent tasks are executed by identical processors (the slaves) under the supervision of a special processor (the master)

Heterogeneous version Computing times and communication times are different from slave to slave

Master-slave platform

Master-slave tasking Simple yet efficient

Standard implementation Independent tasks are executed by identical processors (the slaves) under the supervision of a special processor (the master)

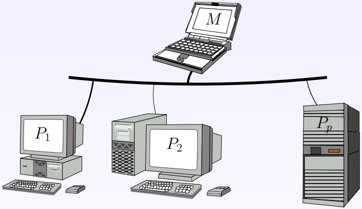


Heterogeneous version Computing times and communication times are different from slave to slave

Master-slave platform

Master-slave tasking Simple yet efficient

Standard implementation Independent tasks are executed by identical processors (the slaves) under the supervision of a special processor (the master)



Heterogeneous version Computing times and communication times are different from slave to slave

Model

- Set of independent tasks to be executed by p slaves
- All tasks are identical: each represents the same amount of computations
- Need d_i time-units to transfer a task from M to P_i, and w_i time-units to execute it on P_i
- Communications obey the one-port model: M can only send one task at a given time-step
- Overlap computations and communications

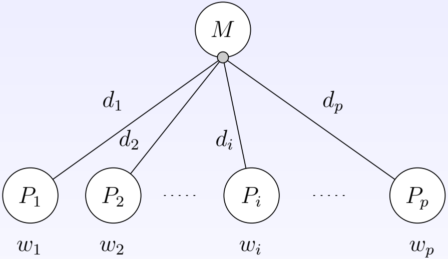
Model

- Set of independent tasks to be executed by p slaves
- All tasks are identical: each represents the same amount of computations

- Need d_i time-units to transfer a task from M to P_i, and w_i time-units to execute it on P_i
- Communications obey the one-port model: M can only send one task at a given time-step
- Overlap computations and communications

Model

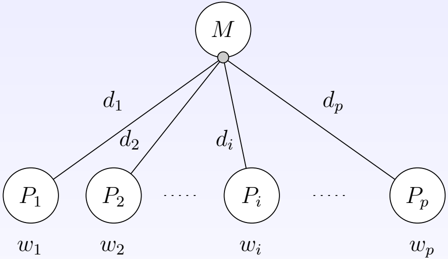
- Set of independent tasks to be executed by p slaves
- All tasks are identical: each represents the same amount of computations



- Need d_i time-units to transfer a task from M to P_i, and w_i time-units to execute it on P_i
- Communications obey the one-port model: M can only send one task at a given time-step
- Overlap computations and communications

Model

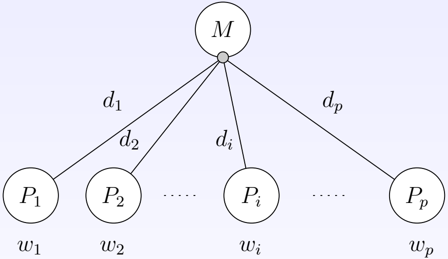
- Set of independent tasks to be executed by p slaves
- All tasks are identical: each represents the same amount of computations



- Need d_i time-units to transfer a task from M to P_i, and w_i time-units to execute it on P_i
- Communications obey the one-port model: M can only send one task at a given time-step
- Overlap computations and communications

Model

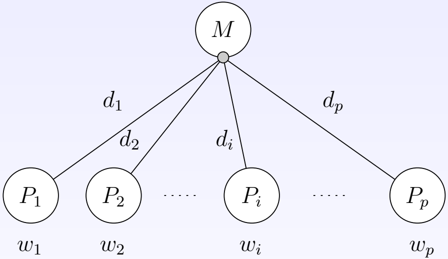
- Set of independent tasks to be executed by p slaves
- All tasks are identical: each represents the same amount of computations



- Need d_i time-units to transfer a task from M to P_i, and w_i time-units to execute it on P_i
- Communications obey the one-port model: M can only send one task at a given time-step
- Overlap computations and communications

Model

- Set of independent tasks to be executed by p slaves
- All tasks are identical: each represents the same amount of computations



- Need d_i time-units to transfer a task from M to P_i, and w_i time-units to execute it on P_i
- Communications obey the one-port model: M can only send one task at a given time-step
- Overlap computations and communications

Complexity results

Definition MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$): Given a master-slave platform with parameters $(d_1, w_1), \ldots, (d_p, w_p)$, what it the minimum time to process n tasks?

MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$) can be solved at cost $O(n^2p^2)$ by a complicated greedy algorithm

If the interconnection network is a linear chain or a harpoon, problem still polynomial

However, for tree-shaped platforms, problem becomes NP-complete

Complexity results

Definition MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$): Given a master-slave platform with parameters $(d_1, w_1), \ldots, (d_p, w_p)$, what it the minimum time to process n tasks?

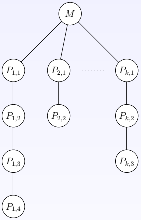
MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$) can be solved at cost $O(n^2 p^2)$ by a complicated greedy algorithm

If the interconnection network is a linear chain or a harpoon, problem still polynomial
However, for tree-shaped platforms, problem becomes NP-complete

Complexity results

Definition MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$): Given a master-slave platform with parameters $(d_1, w_1), \ldots, (d_p, w_p)$, what it the minimum time to process n tasks?

MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$) can be solved at cost $O(n^2p^2)$ by a complicated greedy algorithm



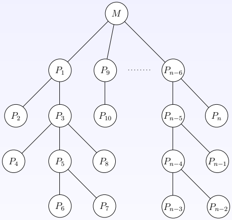
If the interconnection network is a linear chain or a harpoon, problem still polynomial

However, for tree-shaped platforms, problem becomes NP-complete

Complexity results

Definition MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$):
Given a master-slave platform with parameters $(d_1, w_1), \ldots, (d_p, w_p)$, what it the minimum time to process n tasks?

MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$) can be solved at cost $O(n^2p^2)$ by a complicated greedy algorithm



If the interconnection network is a linear chain or a harpoon, problem still polynomial

However, for tree-shaped platforms, problem becomes NP-complete

A model not well-suited. . .

- Hardness comes from the metric: makespan minimization
- Not suited to large-scale distributed platforms
  - Modeling a collection of clusters, and acquiring all various parameters: long, tedious and error-prone
  - Given difficulty and time needed to deploy applications on such platforms, number of tasks expected to be very large
- Concentrate on steady-state, and target complex platforms (with cycles and multiple paths) while designing efficient (asymptotically optimal) schedulings

A model not well-suited. . .

- Hardness comes from the metric: makespan minimization
- Not suited to large-scale distributed platforms
  - Modeling a collection of clusters, and acquiring all various parameters: long, tedious and error-prone
  - Given difficulty and time needed to deploy applications on such platforms, number of tasks expected to be very large
- Concentrate on steady-state, and target complex platforms (with cycles and multiple paths) while designing efficient (asymptotically optimal) schedulings

A model not well-suited. . .

- Hardness comes from the metric: makespan minimization
- Not suited to large-scale distributed platforms
  - Modeling a collection of clusters, and acquiring all various parameters: long, tedious and error-prone
  - Given difficulty and time needed to deploy applications on such platforms, number of tasks expected to be very large
- Concentrate on steady-state, and target complex platforms (with cycles and multiple paths) while designing efficient (asymptotically optimal) schedulings

A model not well-suited. . .

- Hardness comes from the metric: makespan minimization
- Not suited to large-scale distributed platforms
  - Modeling a collection of clusters, and acquiring all various parameters: long, tedious and error-prone
  - Given difficulty and time needed to deploy applications on such platforms, number of tasks expected to be very large
- Concentrate on steady-state, and target complex platforms (with cycles and multiple paths) while designing efficient (asymptotically optimal) schedulings

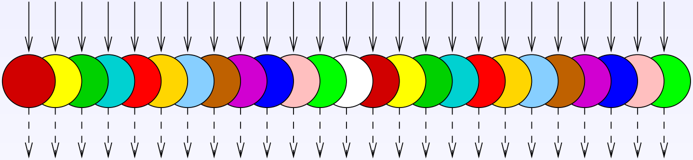
A model not well-suited. . .

- Hardness comes from the metric: makespan minimization
- Not suited to large-scale distributed platforms
  - Modeling a collection of clusters, and acquiring all various parameters: long, tedious and error-prone
  - Given difficulty and time needed to deploy applications on such platforms, number of tasks expected to be very large
- Concentrate on steady-state, and target complex platforms (with cycles and multiple paths) while designing efficient (asymptotically optimal) schedulings

Application graph

n problem instances $\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots, \mathcal{P}^{(n)}$, where n is large

Each problem corresponds to a copy of the same task graph $G_A = (V_A, E_A)$, the application graph

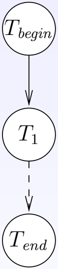


T_{begin} et T_{end} are fictitious tasks, used to model the scattering of input files and the gathering of output files

Application graph

n problem instances $\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots, \mathcal{P}^{(n)}$, where n is large

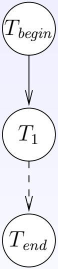
Each problem corresponds to a copy of the same task graph $G_A = (V_A, E_A)$, the application graph



T_{begin} et T_{end} are fictitious tasks, used to model the scattering of input files and the gathering of output files

Application graph

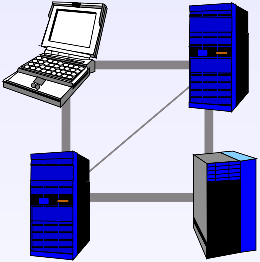
n problem instances $\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots, \mathcal{P}^{(n)}$, where n is large
Each problem corresponds to a copy of the same task graph $G_A = (V_A, E_A)$, the application graph



T_{begin} et T_{end} are fictitious tasks, used to model the scattering of input files and the gathering of output files

Platform graph

Target platform represented by platform graph $G_P = (V_P, E_P)$

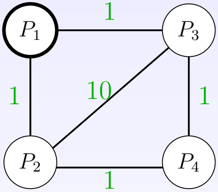


Edge $P_i \to P_j$ is labeled with $c_{i,j}$: time needed to send a unit-length message from P_i to P_j

Communication model: full overlap, one-port for incoming and outgoing messages

Platform graph

Target platform represented by platform graph $G_P = (V_P, E_P)$

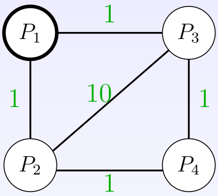


Edge $P_i \rightarrow P_j$ is labeled with $c_{i,j}$: time needed to send a unit-length message from P_i to P_j

Communication model: full overlap, one-port for incoming and outgoing messages

Platform graph

Target platform represented by platform graph $G_P = (V_P, E_P)$



Edge $P_i \rightarrow P_j$ is labeled with $c_{i,j}$: time needed to send a unit-length message from P_i to P_j

Communication model: full overlap, one-port for incoming and outgoing messages

Computations and communications

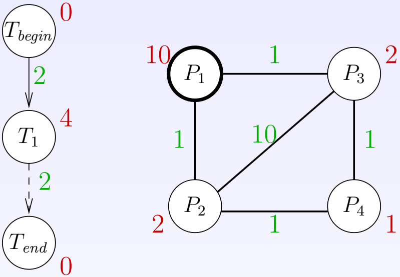
P_i requires $w_{i,k}$ time-units to process task T_k ($k \in \{begin, 1, end\}$).

Edge $e_{k,l} : T_k \rightarrow T_l$ in G_A is labeled with $data_{k,l}$: data volume generated by T_k and used by T_l

Transfer time of a file $e_{k,l}$ from P_i to P_j: $data_{k,l} \times c_{i,j}$

Computations and communications

P_i requires $w_{i,k}$ time-units to process task T_k ($k \in \{begin, 1, end\}$).

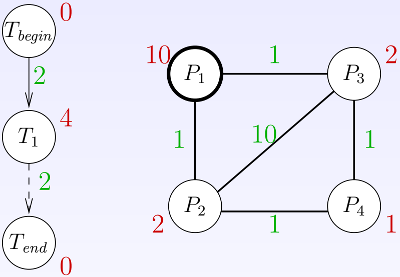


Edge $e_{k,l} : T_k \rightarrow T_l$ in G_A is labeled with $data_{k,l}$: data volume generated by T_k and used by T_l

Transfer time of a file $e_{k,l}$ from P_i to P_j: $data_{k,l} \times c_{i,j}$

Computations and communications

P_i requires $w_{i,k}$ time-units to process task T_k ($k \in \{begin, 1, end\}$).

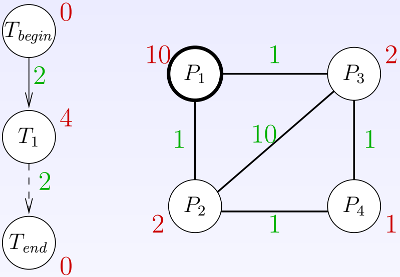


Edge $e_{k,l} : T_k \to T_l$ in G_A is labeled with $data_{k,l}$: data volume generated by T_k and used by T_l

Transfer time of a file $e_{k,l}$ from P_i to P_j: $data_{k,l} \times c_{i,j}$

Computations and communications

P_i requires $w_{i,k}$ time-units to process task T_k ($k \in \{begin, 1, end\}$).



Edge $e_{k,l} : T_k \rightarrow T_l$ in G_A is labeled with $data_{k,l}$: data volume generated by T_k and used by T_l

Transfer time of a file $e_{k,l}$ from P_i to P_j: $data_{k,l} \times c_{i,j}$

Definitions

Allocation An allocation is a pair of mappings: $\pi : V_A \mapsto V_P$ and $\sigma : E_A \mapsto \{\text{paths in } G_P\}$

Schedule A schedule associated to an allocation (π, σ) is a pair of mappings: $t_\pi : V_A \mapsto \mathbb{R}$ and application $t_\sigma : E_A \times E_P \mapsto \mathbb{R}$, satisfying to:

- precedence constraints
- resource constraints on processors
- resource constraints on network links
- one-port constraints

Definitions

Allocation An allocation is a pair of mappings: $\pi : V_A \mapsto V_P$ and $\sigma : E_A \mapsto \{\text{paths in } G_P\}$

Schedule A schedule associated to an allocation (π, σ) is a pair of mappings: $t_\pi : V_A \mapsto \mathbb{R}$ and application $t_\sigma : E_A \times E_P \mapsto \mathbb{R}$, satisfying to:

- precedence constraints
- resource constraints on processors
- resource constraints on network links
- one-port constraints

Activity variables

$cons(P_i, T_k)$: average number of tasks of type T_k processed by P_i every time-unit

$$\forall P_i, \forall T_k \in V_A,\ 0 \leq cons(P_i, T_k) \times w_{i,k} \leq 1$$

$sent(P_i \to P_j, e_{k,l})$: average number of files of type $e_{k,l}$ sent from P_i to P_j every time-unit

$$\forall P_i, P_j,\ 0 \leq sent(P_i \to P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j}) \leq 1$$

Activity variables

$cons(P_i, T_k)$: average number of tasks of type T_k processed by P_i every time-unit

$$\forall P_i, \forall T_k \in V_A,\ 0 \leq cons(P_i, T_k) \times w_{i,k} \leq 1$$

$sent(P_i \rightarrow P_j, e_{k,l})$: average number of files of type $e_{k,l}$ sent from P_i to P_j every time-unit

$$\forall P_i, P_j,\ 0 \leq sent(P_i \rightarrow P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j}) \leq 1$$

Steady-state equations

One-port for outgoing communications P_i sends messages to its neighbors sequentially

$$\forall P_i, \sum_{P_i \to P_j} \sum_{e_{k,l} \in E_A} \left(sent(P_i \to P_j, e_{k,l}) \times data_{k,l} \times c_{i,j} \right) \leq 1$$

One-port for ingoing communications P_i receives messages sequentially

$$\forall P_i, \sum_{P_j \to P_i} \sum_{e_{k,l} \in E_A} \left(sent(P_j \to P_i, e_{k,l}) \times data_{k,l} \times c_{j,i} \right) \leq 1$$

Overlap Computations and communications take place simultaneously

$$\forall P_i, \sum_{T_k \in V_A} cons(P_i, T_k) \times w_{i,k} \leq 1$$

Steady-state equations

One-port for outgoing communications P_i sends messages to its neighbors sequentially

$$\forall P_i, \sum_{P_i \to P_j} \sum_{e_{k,l} \in E_A} \left(sent(P_i \to P_j, e_{k,l}) \times data_{k,l} \times c_{i,j} \right) \leq 1$$

One-port for ingoing communications P_i receives messages sequentially

$$\forall P_i, \sum_{P_j \to P_i} \sum_{e_{k,l} \in E_A} \left(sent(P_j \to P_i, e_{k,l}) \times data_{k,l} \times c_{j,i} \right) \leq 1$$

Overlap Computations and communications take place simultaneously

$$\forall P_i, \sum_{T_k \in V_A} cons(P_i, T_k) \times w_{i,k} \leq 1$$

Steady-state equations

One-port for outgoing communications P_i sends messages to its neighbors sequentially

$$\forall P_i, \sum_{P_i \to P_j} \sum_{e_{k,l} \in E_A} \left(sent(P_i \to P_j, e_{k,l}) \times data_{k,l} \times c_{i,j} \right) \leq 1$$

One-port for ingoing communications P_i receives messages sequentially

$$\forall P_i, \sum_{P_j \to P_i} \sum_{e_{k,l} \in E_A} \left(sent(P_j \to P_i, e_{k,l}) \times data_{k,l} \times c_{j,i} \right) \leq 1$$

Overlap Computations and communications take place simultaneously

$$\forall P_i, \sum_{T_k \in V_A} cons(P_i, T_k) \times w_{i,k} \leq 1$$

Conservation law

Consider a processor P_i and an edge $e_{k,l}$ of the application graph:

Files of type $e_{k,l}$ received: $\displaystyle\sum_{P_j \to P_i} sent(P_j \to P_i, e_{k,l})$

Files of type $e_{k,l}$ generated: $cons(P_i, T_k)$

Files of type $e_{k,l}$ consumed: $cons(P_i, T_l)$

Files of type $e_{k,l}$ sent: $\displaystyle\sum_{P_i \to P_j} sent(P_i \to P_j, e_{k,l})$

In steady state:

$$\forall P_i, \forall e_{k,l} : T_k \to T_l \in E_A,$$

$$\sum_{P_j \to P_i} sent(P_j \to P_i, e_{k,l}) + cons(P_i, T_k) = \sum_{P_i \to P_j} sent(P_i \to P_j, e_{k,l}) + cons(P_i, T_l)$$

Upper bound for the throughput

$$\textsc{Maximize}\ \rho = \sum_{i=1}^{p} cons(P_i, T_{end}),$$

UNDER THE CONSTRAINTS

$$\left\{\begin{array}{ll}
\text{(1a)} & \forall P_i, \forall T_k \in V_A,\ 0 \le cons(P_i, T_k) \times w_{i,k} \le 1 \\
\text{(1b)} & \forall P_i, P_j,\ 0 \le sent(P_i \to P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j}) \le 1 \\
\text{(1c)} & \forall P_i,\ \displaystyle\sum_{P_i \to P_j} \sum_{e_{k,l} \in E_A} \left(sent(P_i \to P_j, e_{k,l}) \times data_{k,l} \times c_{i,j}\right) \le 1 \\
\text{(1d)} & \forall P_i,\ \displaystyle\sum_{P_j \to P_i} \sum_{e_{k,l} \in E_A} \left(sent(P_j \to P_i, e_{k,l}) \times data_{k,l} \times c_{j,i}\right) \le 1 \\
\text{(1e)} & \forall P_i,\ \displaystyle\sum_{T_k \in V_A} cons(P_i, T_k) \times w_{i,k} \le 1 \\
\text{(1f)} & \forall P_i, \forall e_{k,l} \in E_A : T_k \to T_l, \\
& \qquad \displaystyle\sum_{P_j \to P_i} sent(P_j \to P_i, e_{k,l}) + cons(P_i, T_k) = \\
& \qquad\qquad \displaystyle\sum_{P_i \to P_j} sent(P_i \to P_j, e_{k,l}) + cons(P_i, T_l)
\end{array}\right.$$

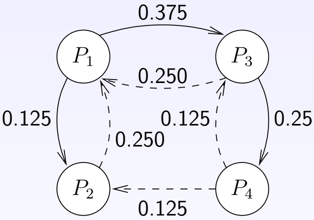
How to design a schedule achieving this throughput?

Back to the example

Computations

	$cons(P_i, T_1)$
P_1	0.025
P_2	0.125
P_3	0.125
P_4	0.250
Total	21 tasks / 40 seconds

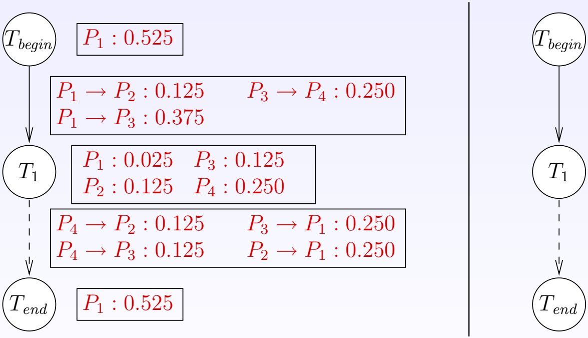
Communications



$$sent(P_i \rightarrow P_j, e_{k,l})$$

Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



$P_1 : 0.525$

$P_1 \to P_2 : 0.125$ $P_3 \to P_4 : 0.250$
$P_1 \to P_3 : 0.375$

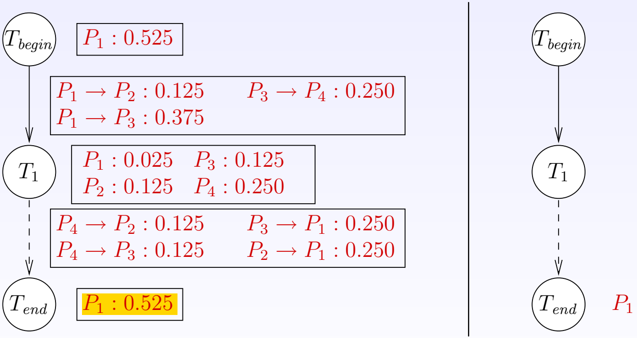
$P_1 : 0.025$ $P_3 : 0.125$
$P_2 : 0.125$ $P_4 : 0.250$

$P_4 \to P_2 : 0.125$ $P_3 \to P_1 : 0.250$
$P_4 \to P_3 : 0.125$ $P_2 \to P_1 : 0.250$

$P_1 : 0.525$

Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



$P_1 : 0.525$

$P_1 \rightarrow P_2 : 0.125$ $P_3 \rightarrow P_4 : 0.250$
$P_1 \rightarrow P_3 : 0.375$

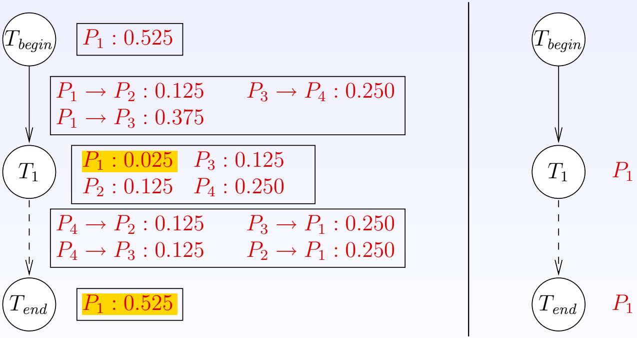
$P_1 : 0.025$ $P_3 : 0.125$
$P_2 : 0.125$ $P_4 : 0.250$

$P_4 \rightarrow P_2 : 0.125$ $P_3 \rightarrow P_1 : 0.250$
$P_4 \rightarrow P_3 : 0.125$ $P_2 \rightarrow P_1 : 0.250$

$P_1 : 0.525$

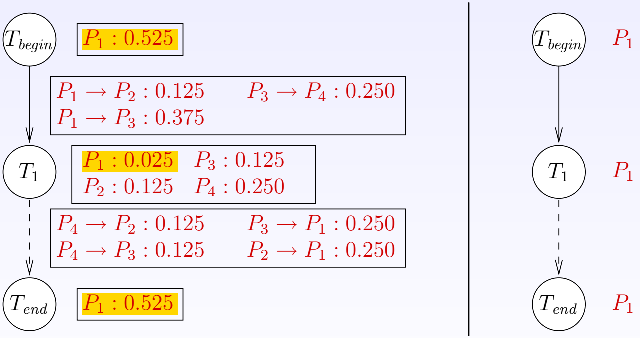
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



$P_1 : 0.525$

$P_1 \rightarrow P_2 : 0.125$ $P_3 \rightarrow P_4 : 0.250$
$P_1 \rightarrow P_3 : 0.375$

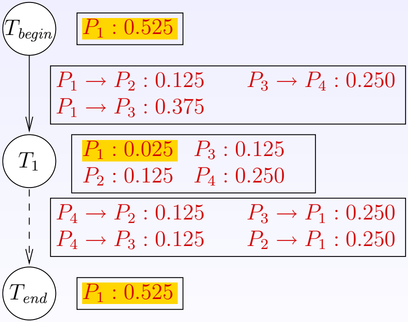
$P_1 : 0.025$ $P_3 : 0.125$
$P_2 : 0.125$ $P_4 : 0.250$

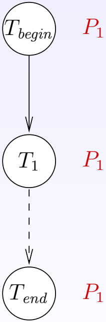
$P_4 \rightarrow P_2 : 0.125$ $P_3 \rightarrow P_1 : 0.250$
$P_4 \rightarrow P_3 : 0.125$ $P_2 \rightarrow P_1 : 0.250$

$P_1 : 0.525$

Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations

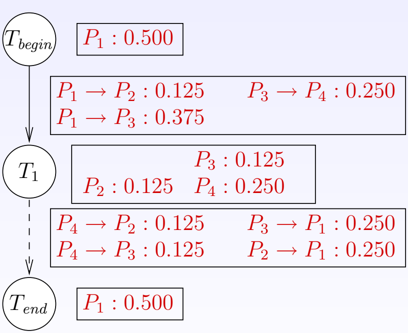


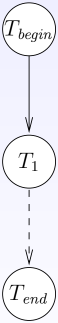


$\mathcal{A}_1 : 0.025$

Decomposition into a set of allocations (1/2)

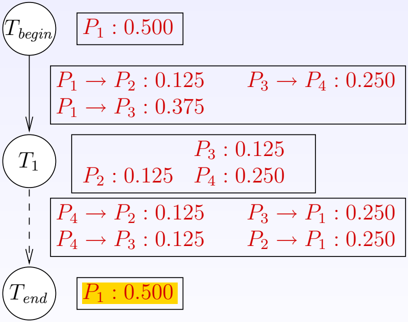
Steady state = superposition of several allocations





Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



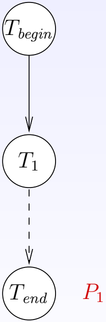
$P_1 : 0.500$

$P_1 \rightarrow P_2 : 0.125$ $P_3 \rightarrow P_4 : 0.250$
$P_1 \rightarrow P_3 : 0.375$

$P_3 : 0.125$
$P_2 : 0.125$ $P_4 : 0.250$

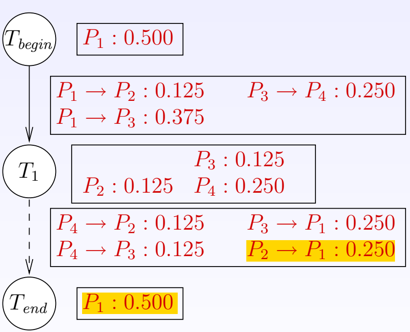
$P_4 \rightarrow P_2 : 0.125$ $P_3 \rightarrow P_1 : 0.250$
$P_4 \rightarrow P_3 : 0.125$ $P_2 \rightarrow P_1 : 0.250$

$P_1 : 0.500$



Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



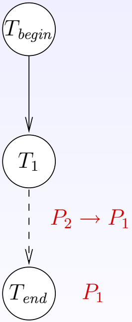
$P_1 : 0.500$

$P_1 \rightarrow P_2 : 0.125$ $P_3 \rightarrow P_4 : 0.250$
$P_1 \rightarrow P_3 : 0.375$

$P_3 : 0.125$
$P_2 : 0.125$ $P_4 : 0.250$

$P_4 \rightarrow P_2 : 0.125$ $P_3 \rightarrow P_1 : 0.250$
$P_4 \rightarrow P_3 : 0.125$ $P_2 \rightarrow P_1 : 0.250$

$P_1 : 0.500$

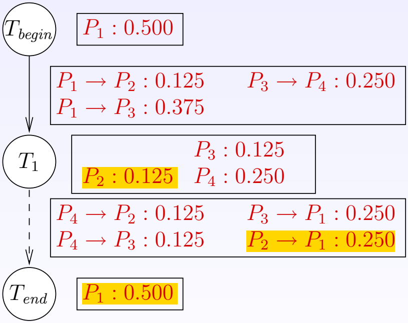


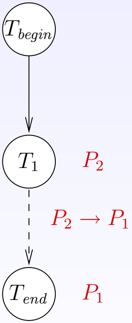
$P_2 \rightarrow P_1$

P_1

Decomposition into a set of allocations (1/2)

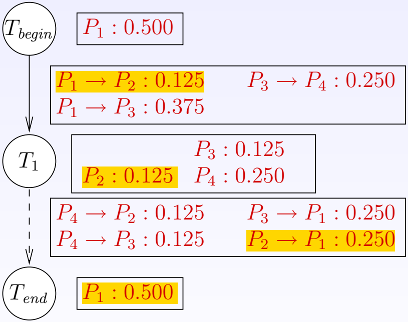
Steady state = superposition of several allocations





Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



T_{begin}

$P_1 : 0.500$

$P_1 \rightarrow P_2 : 0.125$ $P_3 \rightarrow P_4 : 0.250$

$P_1 \rightarrow P_3 : 0.375$

T_1

$P_3 : 0.125$

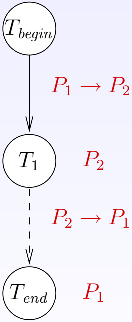
$P_2 : 0.125$ $P_4 : 0.250$

$P_4 \rightarrow P_2 : 0.125$ $P_3 \rightarrow P_1 : 0.250$

$P_4 \rightarrow P_3 : 0.125$ $P_2 \rightarrow P_1 : 0.250$

T_{end}

$P_1 : 0.500$



T_{begin}

$P_1 \rightarrow P_2$

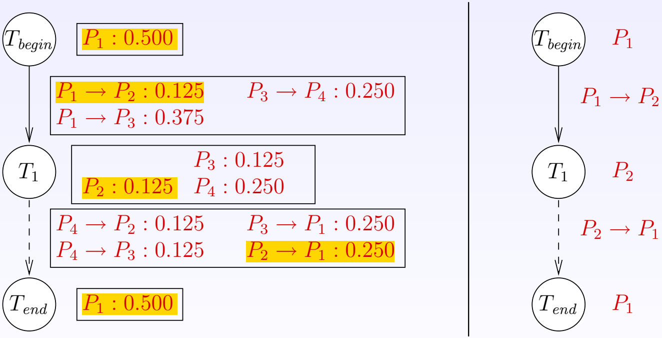
T_1 P_2

$P_2 \rightarrow P_1$

T_{end} P_1

Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



$P_1 : 0.500$

$P_1 \rightarrow P_2 : 0.125$ $P_3 \rightarrow P_4 : 0.250$
$P_1 \rightarrow P_3 : 0.375$

$P_3 : 0.125$
$P_2 : 0.125$ $P_4 : 0.250$

$P_4 \rightarrow P_2 : 0.125$ $P_3 \rightarrow P_1 : 0.250$
$P_4 \rightarrow P_3 : 0.125$ $P_2 \rightarrow P_1 : 0.250$

$P_1 : 0.500$

P_1

$P_1 \rightarrow P_2$

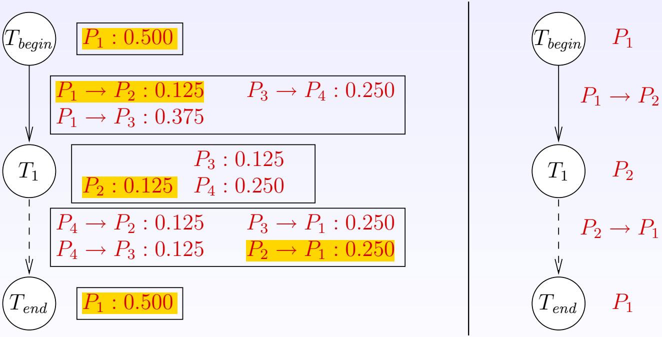
P_2

$P_2 \rightarrow P_1$

P_1

Decomposition into a set of allocations (1/2)

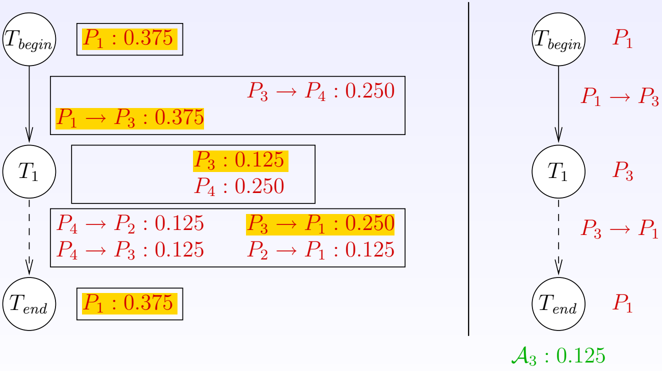
Steady state = superposition of several allocations



$\mathcal{A}_2 : 0.125$

Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



T_{begin}

$P_1 : 0.375$

$P_3 \rightarrow P_4 : 0.250$

$P_1 \rightarrow P_3 : 0.375$

T_1

$P_3 : 0.125$

$P_4 : 0.250$

$P_4 \rightarrow P_2 : 0.125$ $P_3 \rightarrow P_1 : 0.250$

$P_4 \rightarrow P_3 : 0.125$ $P_2 \rightarrow P_1 : 0.125$

T_{end}

$P_1 : 0.375$

T_{begin} P_1

$P_1 \rightarrow P_3$

T_1 P_3

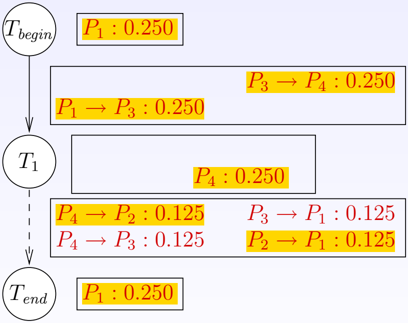
$P_3 \rightarrow P_1$

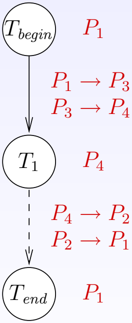
T_{end} P_1

$\mathcal{A}_3 : 0.125$

Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations

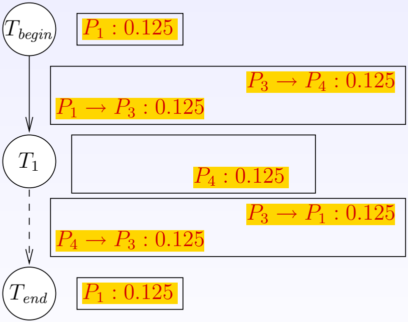


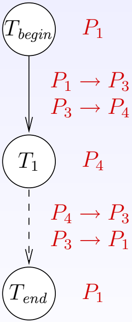


$\mathcal{A}_4 : 0.125$

Decomposition into a set of allocations (1/2)

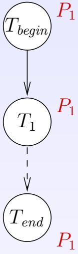
Steady state = superposition of several allocations





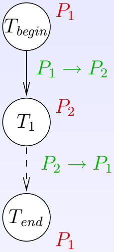
$\mathcal{A}_5 : 0.125$

Decomposition into a set of allocations (2/2)



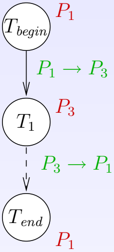
$\mathcal{A}_1$

0,025



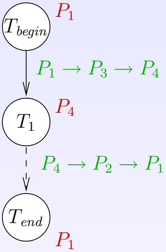
$\mathcal{A}_2$

0,125



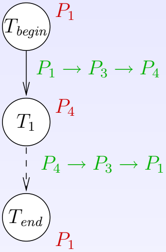
$\mathcal{A}_3$

0,125



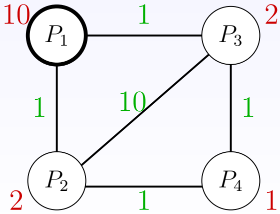
$\mathcal{A}_4$

0,125

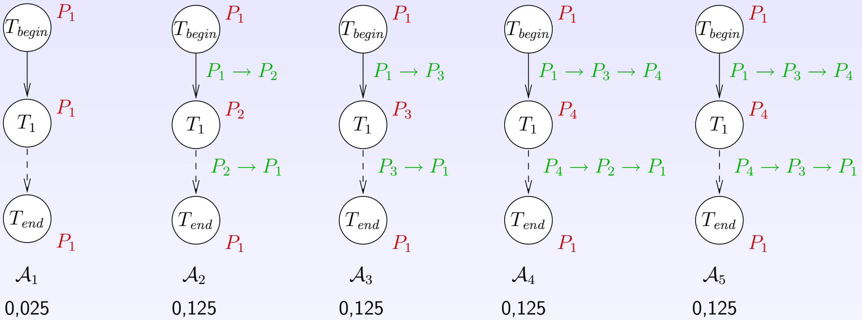


$\mathcal{A}_5$

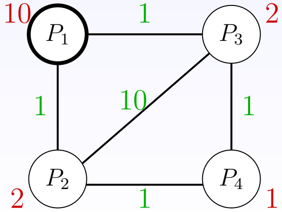
0,125



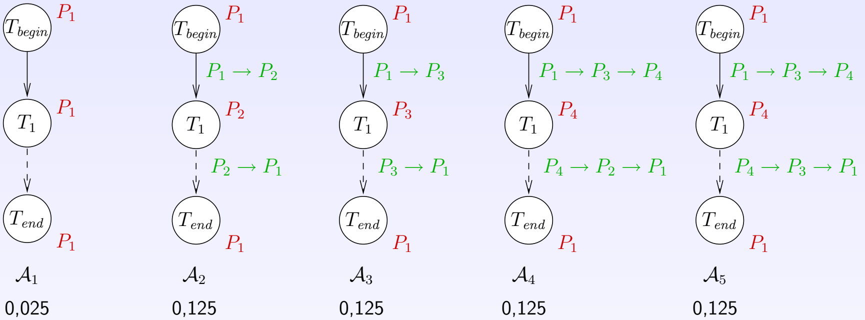
Decomposition into a set of allocations (2/2)



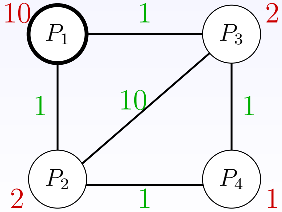
This decomposition is always possible



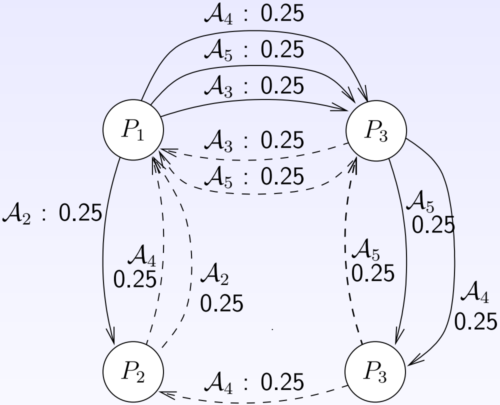
Decomposition into a set of allocations (2/2)



How to orchestrate these allocations?

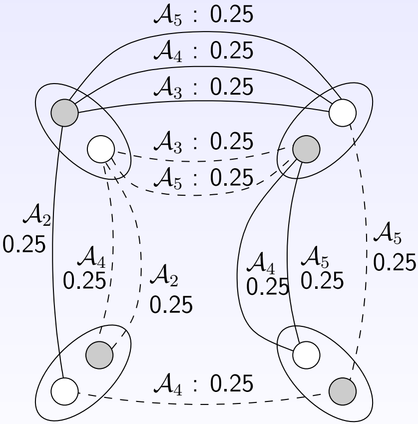


Communication graph

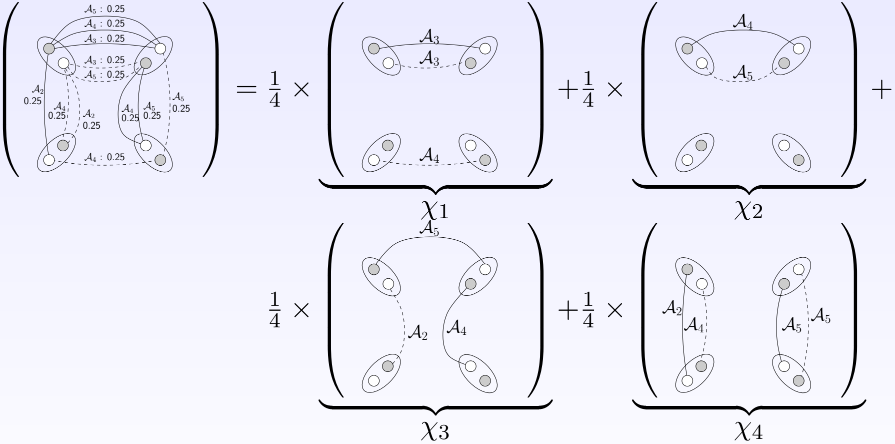


Fraction of time spent transferring some $e_{k,l}$ file from P_i to P_j for a given allocation

One-port constraints = matching

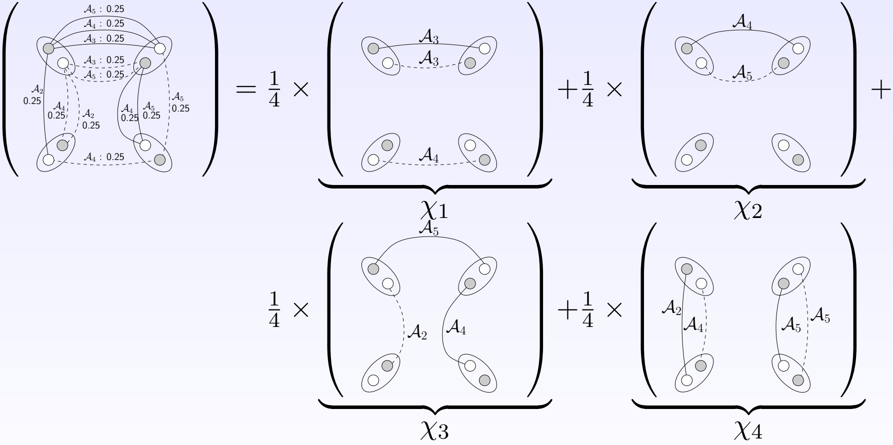


Edge coloring (decomposition into matchings)



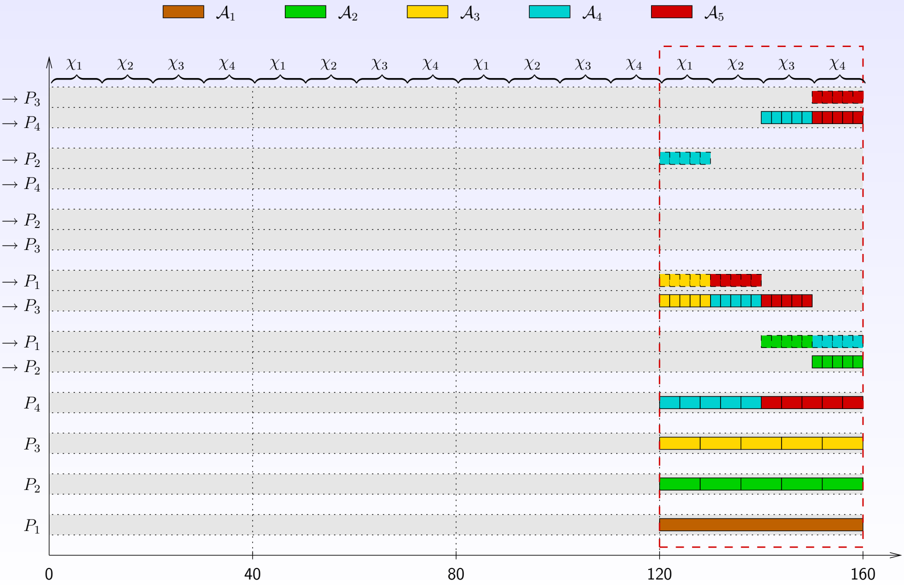
This decomposition is always possible

Edge coloring (decomposition into matchings)

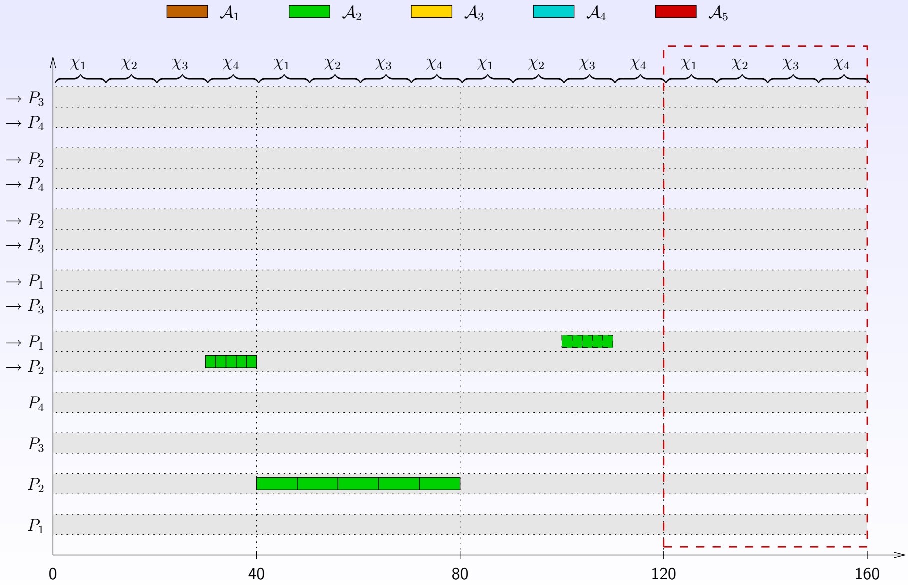


This decomposition is always possible

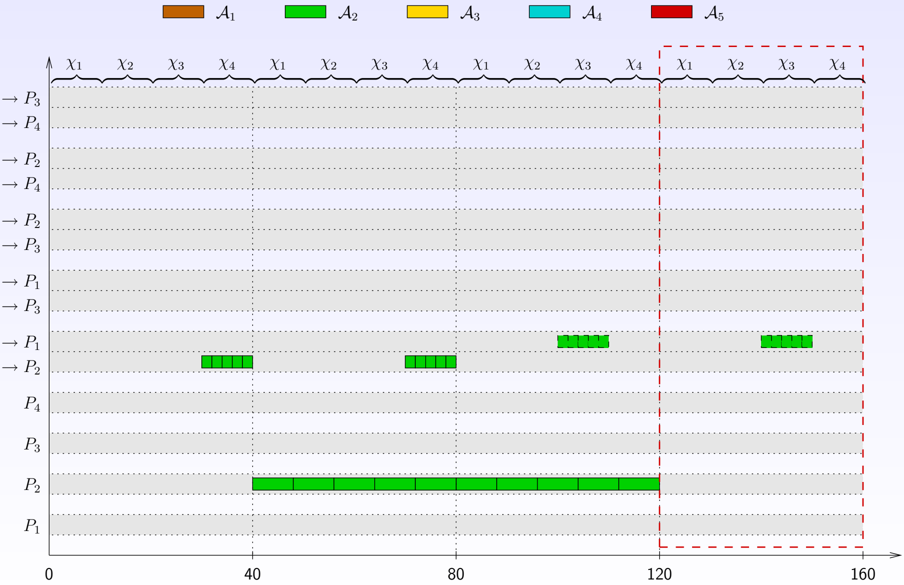
Cyclic scheduling achieving optimal throughput



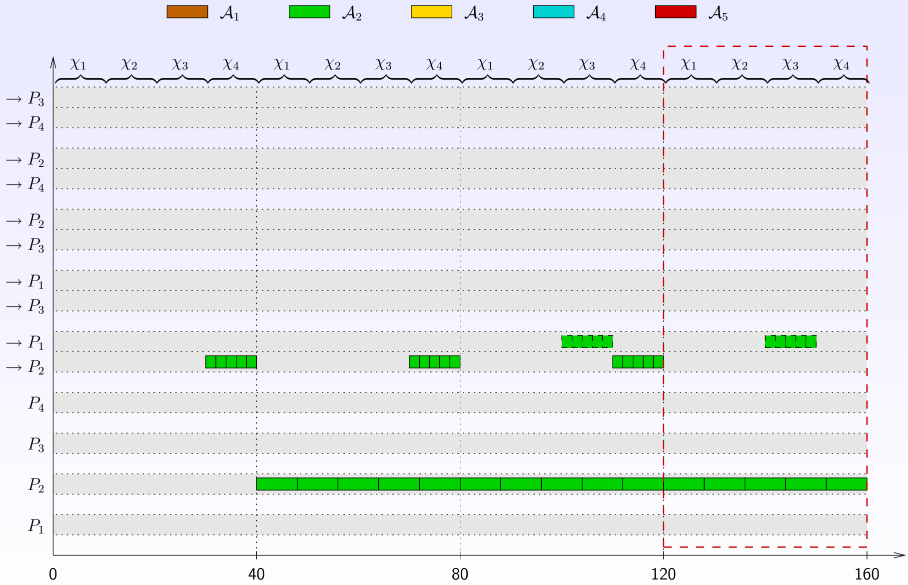
Cyclic scheduling achieving optimal throughput



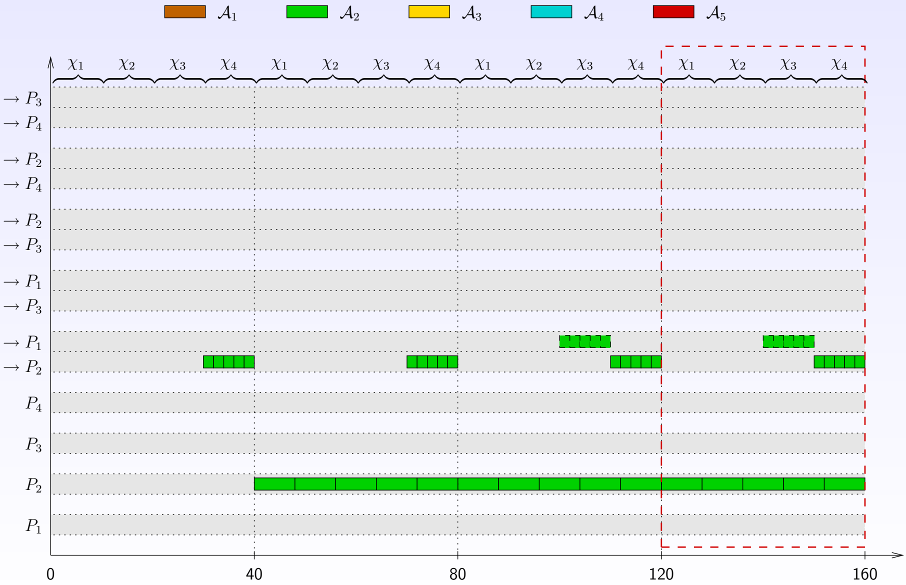
Cyclic scheduling achieving optimal throughput



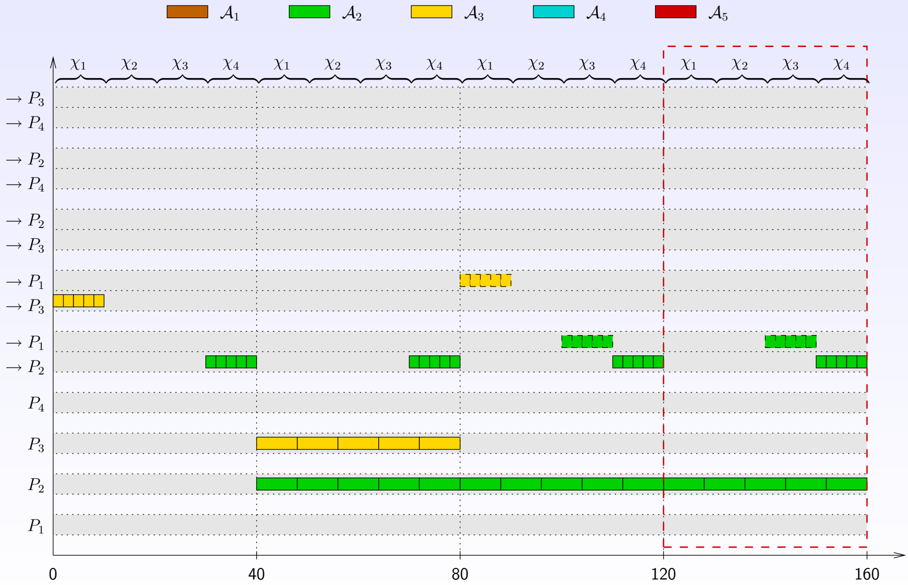
Cyclic scheduling achieving optimal throughput



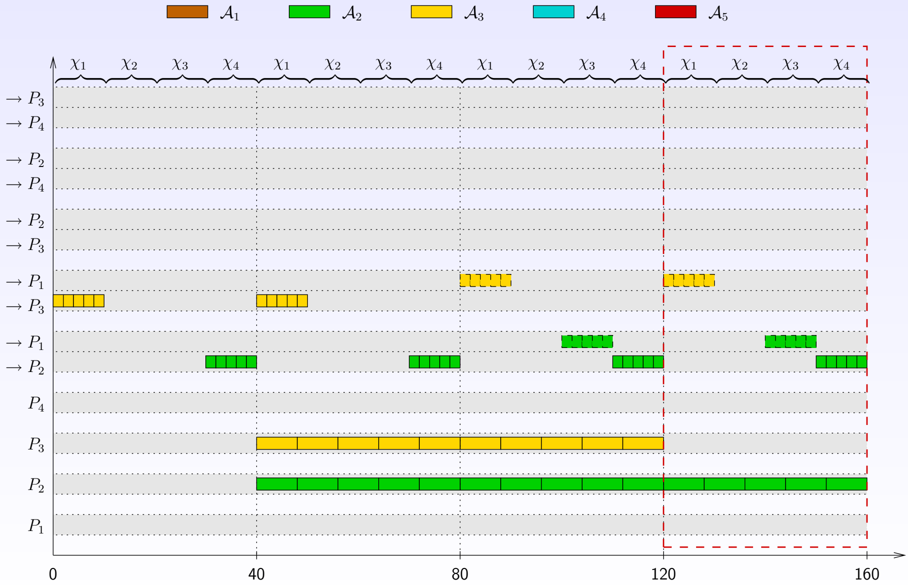
Cyclic scheduling achieving optimal throughput



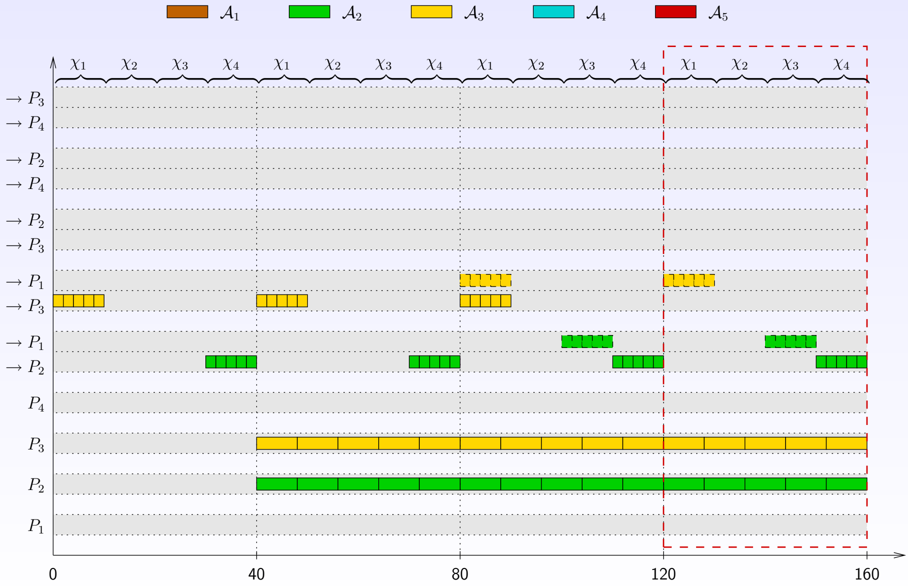
Cyclic scheduling achieving optimal throughput



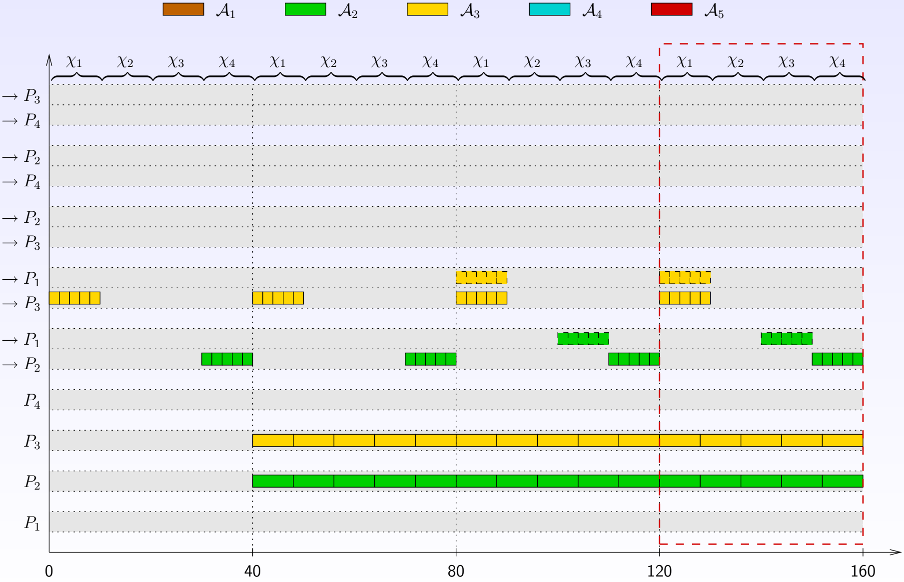
Cyclic scheduling achieving optimal throughput



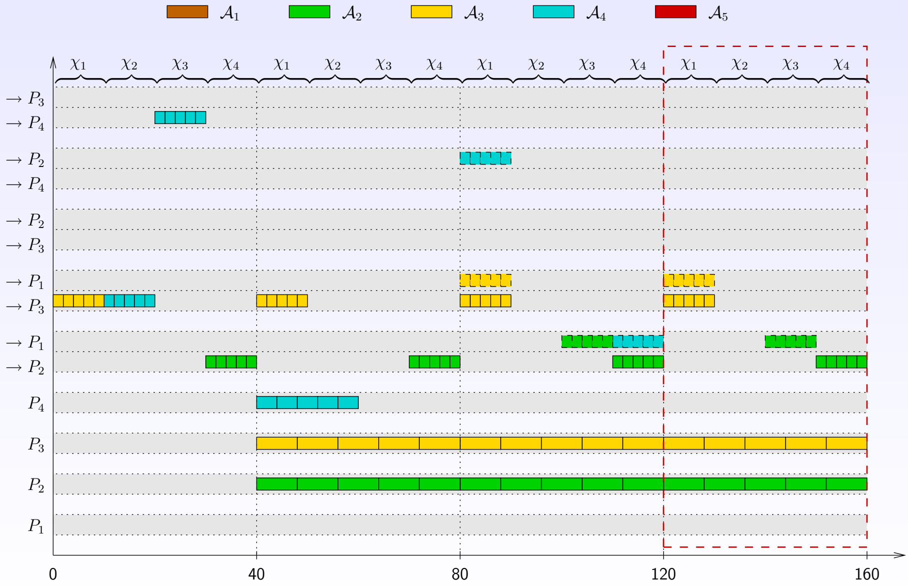
Cyclic scheduling achieving optimal throughput



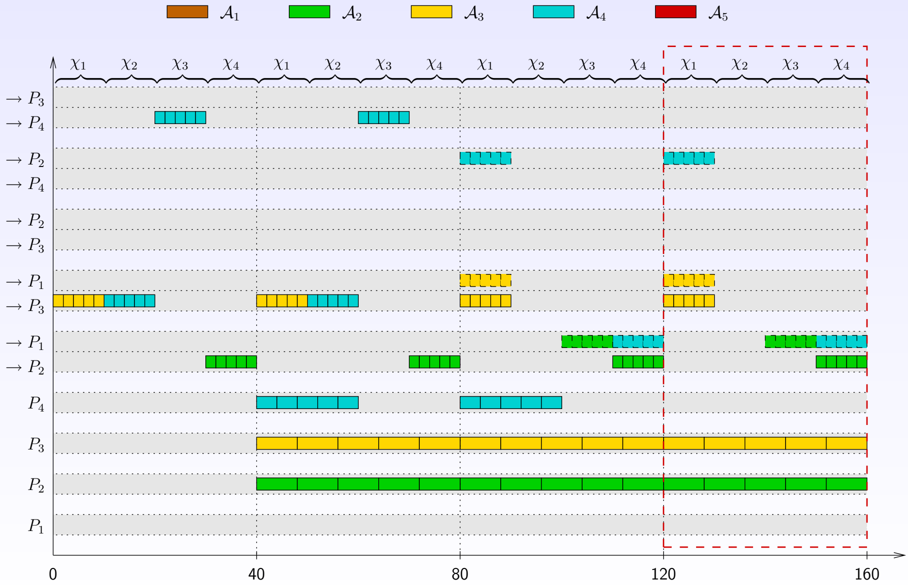
Cyclic scheduling achieving optimal throughput



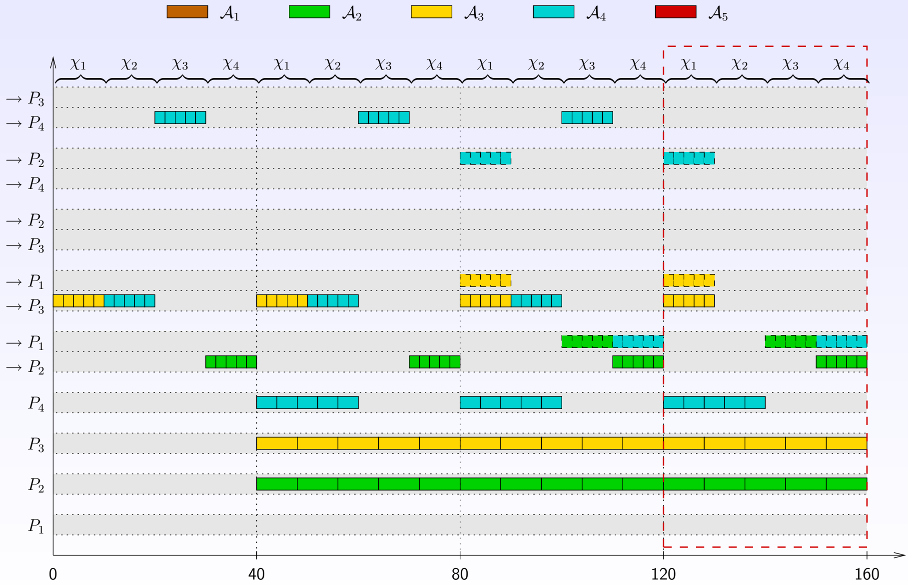
Cyclic scheduling achieving optimal throughput



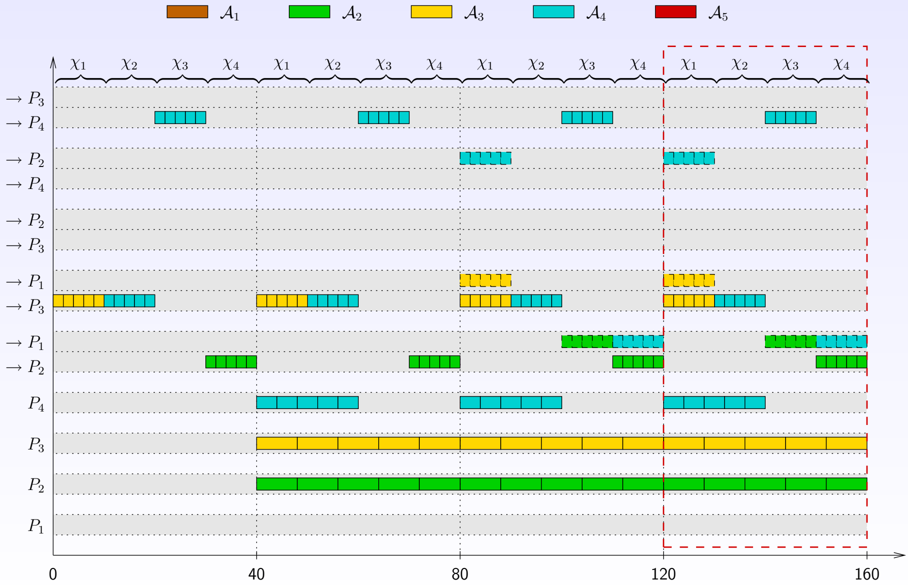
Cyclic scheduling achieving optimal throughput



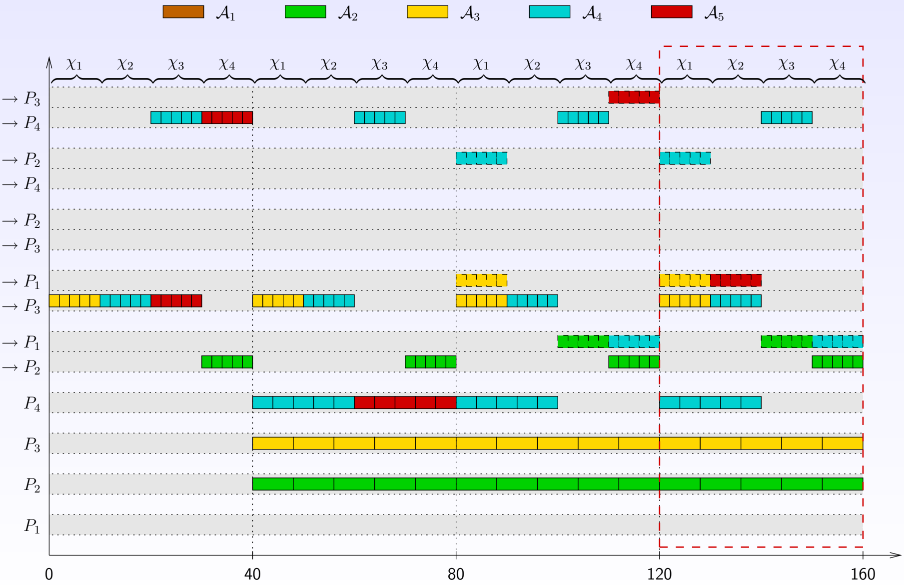
Cyclic scheduling achieving optimal throughput



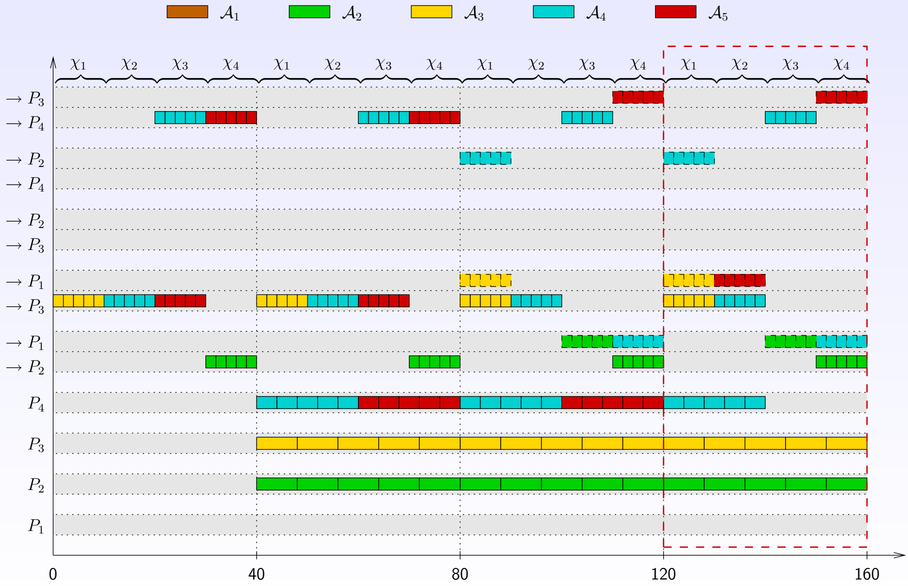
Cyclic scheduling achieving optimal throughput



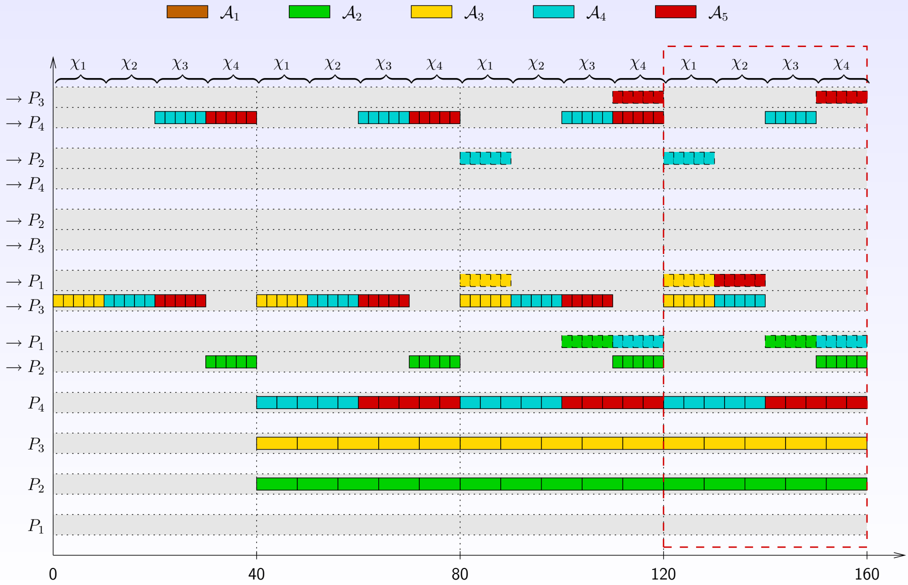
Cyclic scheduling achieving optimal throughput



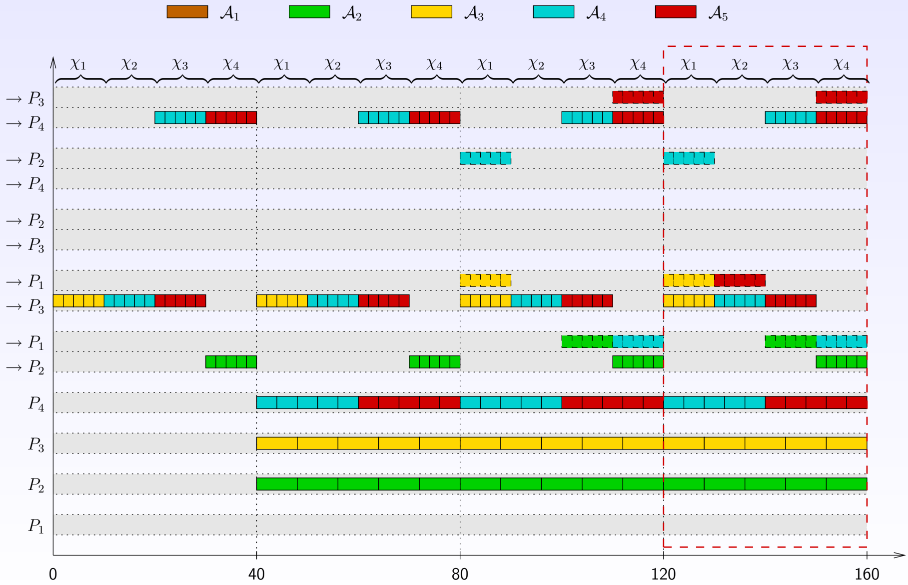
Cyclic scheduling achieving optimal throughput



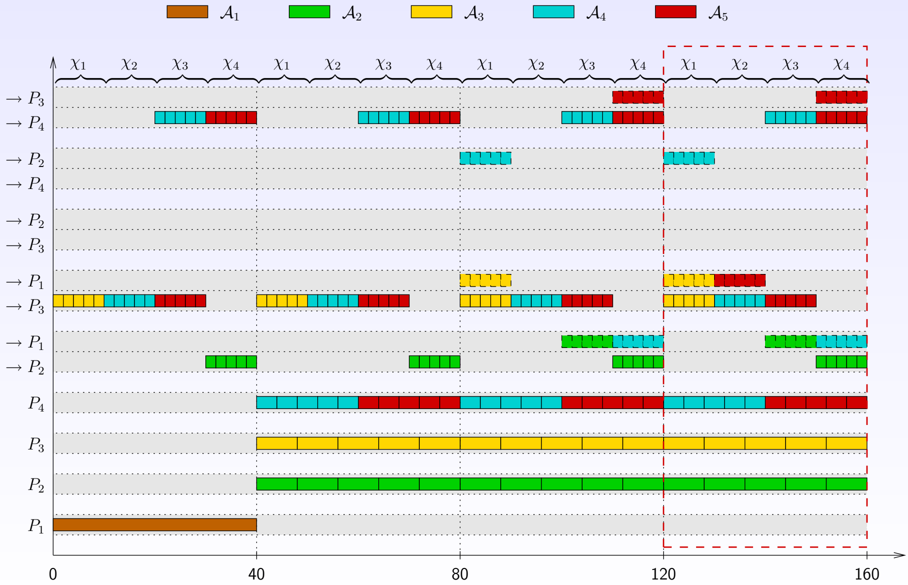
Cyclic scheduling achieving optimal throughput



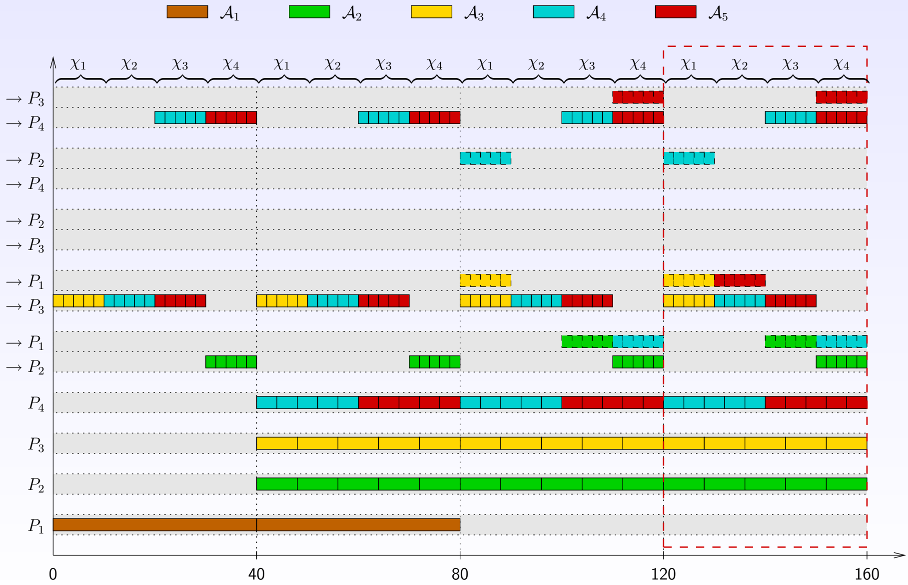
Cyclic scheduling achieving optimal throughput



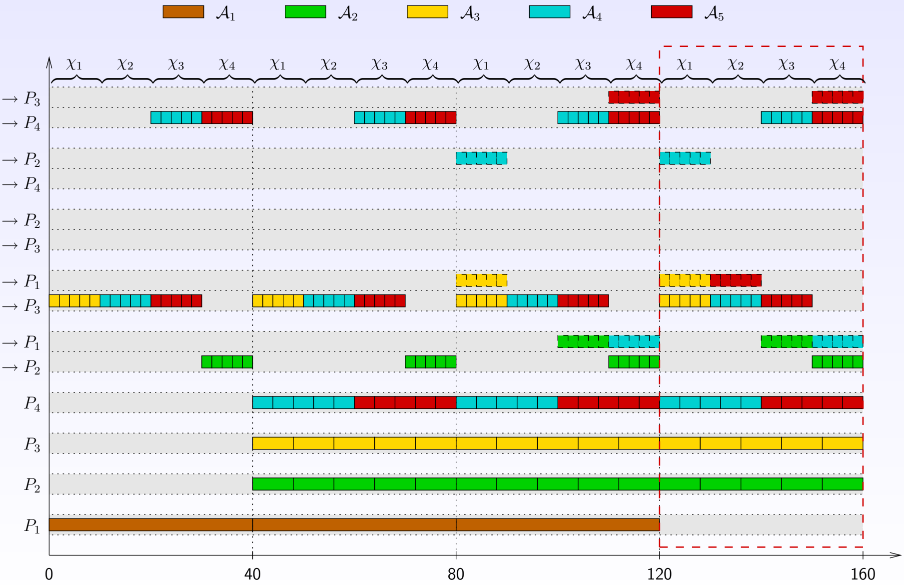
Cyclic scheduling achieving optimal throughput



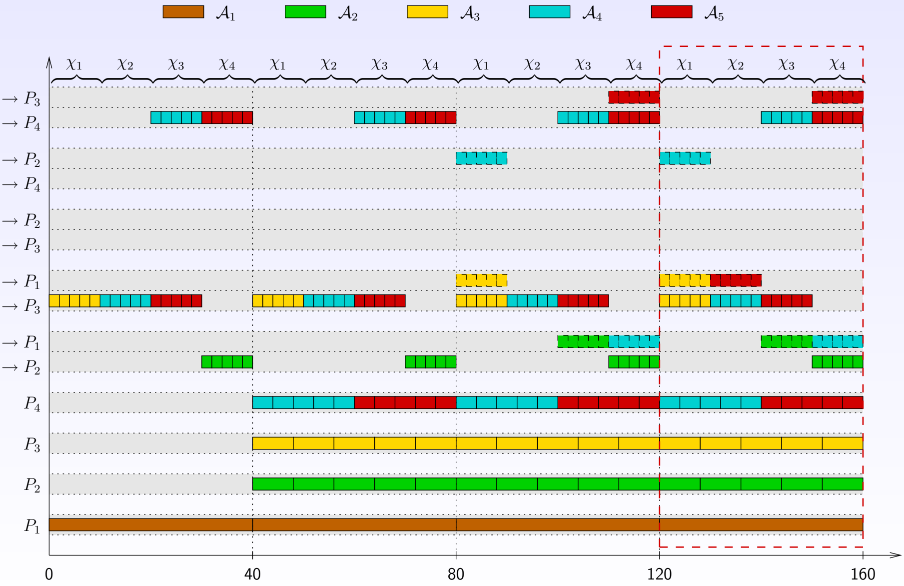
Cyclic scheduling achieving optimal throughput



Cyclic scheduling achieving optimal throughput



Cyclic scheduling achieving optimal throughput



Asymptotically optimal schedule

- The technique used in the example is
  - general
  - polynomial
- The resulting schedule is asymptotically optimal: within T time-steps, it differs from the optimal schedule by a constant number of tasks (independent of T)

Extensions to collections of general task graphs

- More difficult but possible
- Maximizing throughput NP-hard ☹
- Most application DAGs have polynomial number of joins
  $\Rightarrow$ polynomial solution ☺

Perspectives

- Macro-communications (scatter, gather, reduce, broadcast, multicast,. . .)
- Open problems:
    - Period length, approximating cyclic pattern
    - When problem remains difficult after steady-state relaxation?
    - Stability, robustness in front of load variations

Bibliography

- Packet routing:
  Asymptotically optimal algorithms for job shop scheduling and packet routing, D. Bertsimas and D. Gamarnik,
  In *Journal of Algorithms 33, 2 (1999), 296-318*

- Steady-state for independent tasks:
  Scheduling strategies for master-slave tasking on heterogeneous processor platforms, C. Banino et al.,
  In *IEEE TPDS 15, 4 (2004), 319-330*

- Steady-state for DAGs: complete reseach report
  Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogeneous platforms., O. Beaumont et al.,
  LIP research report, RR-2004-20,
  http://www.ens-lyon.fr/LIP/Pub/Rapports/RR/RR2004/RR2004-20.ps.gz

- With bounded multi-port model:
  Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput, B. Hong and V.K. Prasanna,
  In *IEEE IPDPS (2004)*

Outline

1. Background on traditional scheduling
2. Divisible Load Scheduling
3. Steady-State Scheduling
4. Simulation for Grid Computing
    - Validation Problem
    - Platform modeling
    - Simulation
    - SimGrid

Analytical or Experimental validation ?

- Scheduling theory:
  purely analytical / mathematical models for Grid computing
  - makes it possible to prove interesting theorems
  - often too simplistic to convince practitioners
  - but generally useful for understanding principles
- Heterogeneity, latencies, . . . render scheduling problems NP-hard
  - Design low complexity heuristics
  - How to compare two different heuristics ?

$\rightsquigarrow$ Need for experiments

Analytical or Experimental validation ?

- Scheduling theory:
  purely analytical / mathematical models for Grid computing
  - makes it possible to prove interesting theorems
  - often too simplistic to convince practitioners
  - but generally useful for understanding principles
- Heterogeneity, latencies, . . . render scheduling problems NP-hard
  - Design low complexity heuristics
  - How to compare two different heuristics ?

$\leadsto$ Need for experiments

Analytical or Experimental validation ?

- Scheduling theory:
  purely analytical / mathematical models for Grid computing
  - makes it possible to prove interesting theorems
  - often too simplistic to convince practitioners
  - but generally useful for understanding principles
- Heterogeneity, latencies,. . . render scheduling problems NP-hard
  - Design low complexity heuristics
  - How to compare two different heuristics ?

$\rightsquigarrow$ Need for experiments

Grid Experiments (1/3)

- Real-world experiments are good
  - Eminently believable
  - Demonstrates that proposed approach can be implemented in practice

But...

- Can be time-intensive
  Execution of "applications" for hours, days, months,. . .
- Can be labor-intensive
  Entire application needs to be built and functional.
  Is it a good engineering practice to carry out many entire solutions to find out which ones works best?

Grid Experiments (1/3)

- Real-world experiments are good
  - Eminently believable
  - Demonstrates that proposed approach can be implemented in practice

But...

- Can be time-intensive
  Execution of “applications” for hours, days, months,. . .
- Can be labor-intensive
  Entire application needs to be built and functional.
  Is it a good engineering practice to carry out many entire solutions to find out which ones works best?

Grid Experiments (1/3)

- Real-world experiments are good
  - Eminently believable
  - Demonstrates that proposed approach can be implemented in practice

But...

- Can be time-intensive

  Execution of “applications” for hours, days, months,. . .
- Can be labor-intensive

  Entire application needs to be built and functional.

  Is it a good engineering practice to carry out many entire solutions to find out which ones works best?

Grid Experiments (2/3)

What experimental test-bed?

- My own little test-bed
  well-behaved, controlled, stable, often not representative of real Grids.
- Real grid platforms
  - (Still) challenging for many grid researchers to obtain
  - Not built as a tool for my experiments:
    - other user may disrupt my experiments
    - other users may find my experiments disruptive
  - Platform will experience failures
  - Platform configuration may change drastically while experiments are being conducted
  - Experiments are uncontrolled and unrepeatable: even if disruption from other users is part of the experiments, it prevents comparative runs of different heuristics

Grid Experiments (2/3)

What experimental test-bed?

- My own little test-bed
  well-behaved, controlled, stable, often not representative of real Grids.
- Real grid platforms
  - (Still) challenging for many grid researchers to obtain
  - Not built as a tool for my experiments:
    - other user may disrupt my experiments
    - other users may find my experiments disruptive
  - Platform will experience failures
  - Platform configuration may change drastically while experiments are being conducted
  - Experiments are uncontrolled and unrepeatable: even if disruption from other users is part of the experiments, it prevents comparative runs of different heuristics

Grid Experiments (3/3)

$\leadsto$ Difficult to obtain statistically significant results on an appropriate test-bed

And to make things worse...

- Experiments are limited to the test-bed
  - What part of the results are due to idiosyncrasies of the test-bed?
  - Extrapolations are possible, but rarely convincing
- Difficult for others to reproduce results
  This is the basis for scientific advances!

Grid experiments are limited and non reproducible.

Grid Experiments (3/3)

$\rightsquigarrow$ Difficult to obtain statistically significant results on an appropriate test-bed

And to make things worse...

- Experiments are limited to the test-bed
  - What part of the results are due to idiosyncrasies of the test-bed?
  - Extrapolations are possible, but rarely convincing
- Difficult for others to reproduce results
  This is the basis for scientific advances!

Grid experiments are limited and non reproducible.

Grid Experiments (3/3)

$\rightsquigarrow$ Difficult to obtain statistically significant results on an appropriate test-bed

And to make things worse...

- Experiments are limited to the test-bed
  - What part of the results are due to idiosyncrasies of the test-bed?
  - Extrapolations are possible, but rarely convincing
- Difficult for others to reproduce results
  This is the basis for scientific advances!

Grid experiments are limited and non reproducible.

Grid Experiments (3/3)

$\rightsquigarrow$ Difficult to obtain statistically significant results on an appropriate test-bed

And to make things worse...

- Experiments are limited to the test-bed
  - What part of the results are due to idiosyncrasies of the test-bed?
  - Extrapolations are possible, but rarely convincing
- Difficult for others to reproduce results
  This is the basis for scientific advances!

Grid experiments are limited and non reproducible.

Simulation

Simulation can solve many (all) of these difficulties

- No need to build a real system
- Conduct controlled and repeatable experiments
- In principle, no limits to experimental scenarios
- Possible for anybody to reproduce results

Definition (Simulation)

Attempting to predict aspects of the behavior of some system by creating an approximate (mathematical) model of it.

Key question: Validation (correspondence between simulation and real-world)

Simulation

Simulation can solve many (all) of these difficulties

- No need to build a real system
- Conduct controlled and repeatable experiments
- In principle, no limits to experimental scenarios
- Possible for anybody to reproduce results

Definition (Simulation)

Attempting to predict aspects of the behavior of some system by creating an approximate (mathematical) model of it.

Key question: Validation (correspondence between simulation and real-world)

Grid Simulations

Challenges for grid simulations:

- Consider complex network topologies (multi-hop networks, heterogeneous bandwidths and latencies, non-negligible latencies, complex bandwidth sharing behaviors, contention with other traffic)
- Overhead of middleware
- Complex resource access/management policies
- Interference of communication and computation

Two main questions for grid simulations:

1. What does a "representative" Grid look like?
2. How does one do simulation on a synthetic representative Grid?

Grid Simulations

Challenges for grid simulations:

- Consider complex network topologies (multi-hop networks, heterogeneous bandwidths and latencies, non-negligible latencies, complex bandwidth sharing behaviors, contention with other traffic)
- Overhead of middleware
- Complex resource access/management policies
- Interference of communication and computation

Two main questions for grid simulations:

1. What does a "representative" Grid look like?
2. How does one do simulation on a synthetic representative Grid?

Platform modeling

Network modeling

- Depending on the application, clarify the network contention (if any)
- Network topology generators
- Provide link characteristics (bandwidth, latency,. . .)

Computational resources

- Examine existing resources adapted to my application,
- Design generative model, following key characteristics

Resource availability

- Probabilistic models
- Traces (NWS)
- Workload models for batch schedulers

Simulation in a nutshell

Simulations are configurable, repeatable, fast.

Key question: “Validation: correspondence between simulation and real world”.

Simulation in a nutshell

Simulations are configurable, repeatable, fast.
Key question: "Validation: correspondence between simulation and real world".

more abstract	Mathematical Simulation	Based solely on equations
	Discrete-Event Simulation	Abstraction of system as a set of dependant actions and events (fine- or coarse- grain)
less abstract	Emulation	Trapping and virtualization of low-level application/system actions

Simulation in a nutshell

Simulations are configurable, repeatable, fast.
Key question: “Validation: correspondence between simulation and real world”.

		Network
more abstract	Mathematical Simulation	Macroscopic: flows in a pipe (coarse-grain d.e simulation + math. simulation)
	Discrete-Event Simulation	Microscopic: packet-level (fine-grain d.e. simulation)
less abstract	Emulation	Actual flows go through some network

Simulation in a nutshell

Simulations are configurable, repeatable, fast.
Key question: "Validation: correspondence between simulation and real world".

		CPU
more abstract	Mathematical Simulation	Macroscopic: flows in a pipe (coarse-grain d.e simulation + math. simulation)
	Discrete-Event Simulation	Microscopic: Cycle-accurate simulation (fine-grain d.e. simulation)
less abstract	Emulation	Virtualization via another CPU/virtual machine

Simulation in a nutshell

Simulations are configurable, repeatable, fast.
Key question: "Validation: correspondence between simulation and real world".

		Application
more abstract	Mathematical Simulation	Macroscopic: application = analytical "flow"
	Discrete-Event Simulation	Less Macroscopic: set of abstract tasks with resource needs and dependancies
less abstract	Emulation	Virtualization (emulation of actual code with trapping of application generated events)

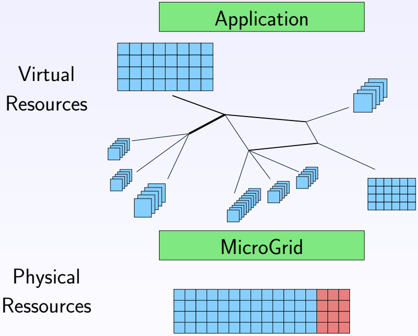
MicroGrid

MicroGrid is a UCSD project lead by Andrew Chien.

Applications are supported by emulation and virtualization: Actual application code is executed on "virtualized" resources

MicroGrid accounts for CPU and network

Resource	gethostnames, sockets, GIS, MDS, NWS are wrapped
CPU	Direct execution on a fraction of physical CPU: find a good mapping
Network	Packet-level simulation (parallel version of MaSSF)
Time	Synchronize real time and virtual time: find the good execution rate



MicroGrid

MicroGrid is a UCSD project lead by Andrew Chien.

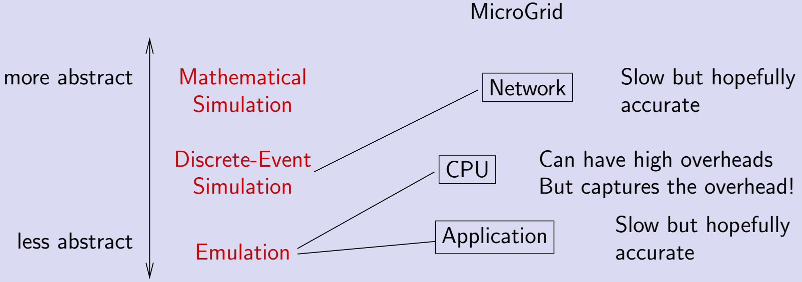
Applications are supported by emulation and virtualization: Actual application code is executed on “virtualized” resources

MicroGrid accounts for CPU and network

Re

N

MicroGrid in a Nutshell



MicroGrid

more abstract

less abstract

Mathematical Simulation

Discrete-Event Simulation

Emulation

Network

CPU

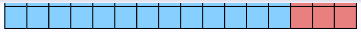
Application

Slow but hopefully accurate

Can have high overheads
But captures the overhead!

Slow but hopefully accurate

virtual time: find the good execution rate



SimGrid

- Originally developed for scheduling research $\rightsquigarrow$ must be fast to allow for thousands of simulation
- Application
  - No real application code is executed
  - Simulation is expressed in term of communicating process
  - Process can perform task communication or computation, described by their resource consumption.
- Resources
  - No virtualization
  - A resource is defined by
    - a rate at which it does "work",
    - a fixed overhead that must be paid by each task,
    - traces of the above if needed + failures.
- Tasks Tasks can use multiple resources
  - data transfer over multiple links,
  - computation that uses a disk and a CPU

SimGrid

- Originally developed for scheduling research $\rightsquigarrow$ must be fast to allow for thousands of simulation
- Application
  - No real application code is executed
  - Simulation is expressed in term of communicating process
  - Process can perform task communication or computation, described by their resource consumption.
- Resources
  - No virtualization
  - A resource is defined by
    - a rate at which it does "work",
    - a fixed overhead that must be paid by each task,
    - traces of the above if needed + failures.
- Tasks Tasks can use multiple resources
  - data transfer over multiple links,
  - computation that uses a disk and a CPU

SimGrid

- Originally developed for scheduling research $\rightsquigarrow$ must be fast to allow for thousands of simulation
- Application
  - No real application code is executed
  - Simulation is expressed in term of communicating process
  - Process can perform task communication or computation, described by their resource consumption.
- Resources
  - No virtualization
  - A resource is defined by
    - a rate at which it does "work",
    - a fixed overhead that must be paid by each task,
    - traces of the above if needed + failures.
- Tasks Tasks can use multiple resources
  - data transfer over multiple links,
  - computation that uses a disk and a CPU

SimGrid

- Originally developed for scheduling research $\rightsquigarrow$ must be fast to allow for thousands of simulation
- Application
  - No real application code is executed
  - Simulation is expressed in term of communicating process
  - Process can perform task communication or computation, described by their resource consumption.
- Resources
  - No virtualization
  - A resource is defined by
    - ★ a rate at which it does "work",
    - ★ a fixed overhead that must be paid by each task,
    - ★ traces of the above if needed + failures.
- Tasks Tasks can use multiple resources
  - data transfer over multiple links,
  - computation that uses a disk and a CPU

SimGrid

- Uses a combination of mathematical simulation and coarse-grain discrete event simulation
  - Simple API to "specify" an application rather than having it already implemented
  - Fast simulation
- Key issue: Resource sharing
  - In MicroGrid: resource sharing "emerges" out of the low level emulation and simulation
    - Packets of different connections interleaved by routers
    - CPU cycles of different processes get slices of the CPU
  - Drawback: slow simulation
  - How can one do something faster that is still reasonable?
  - Come up with macroscopic models of resource sharing

SimGrid

- Uses a combination of mathematical simulation and coarse-grain discrete event simulation
  - Simple API to "specify" an application rather than having it already implemented
  - Fast simulation
- Key issue: Resource sharing
  - In MicroGrid: resource sharing "emerges" out of the low level emulation and simulation
    - Packets of different connections interleaved by routers
    - CPU cycles of different processes get slices of the CPU
  - Drawback: slow simulation
  - How can one do something faster that is still reasonable?
  - Come up with macroscopic models of resource sharing

Resource Sharing in SimGrid

- Resource sharing for CPU:
  - process/threads competing for resource get a fair share of the CPU "in steady state"
  - no need to emulate CPU
  - compute the CPU cycles allocated of each process/thread (rate)
- Resource sharing for the network:
  - many end-points, routers and links,
  - many end-to-end TCP flows ?
  - macroscopic behavior: How much bandwidth does each flow receive?
- Macroscopic TCP modeling:
  - TCP in steady-state implements a type of resource sharing "Max-Min Fairness"
  - Bandwidth allocation can be solved efficiently with appropriate data structure
  - Validated with NS-2 simulators
  - Justified for "long-enough" transfers. . .

Resource Sharing in SimGrid

- Resource sharing for CPU:
  - process/threads competing for resource get a fair share of the CPU "in steady state"
  - no need to emulate CPU
  - compute the CPU cycles allocated of each process/thread (rate)
- Resource sharing for the network:
  - many end-points, routers and links,
  - many end-to-end TCP flows ?
  - macroscopic behavior: How much bandwidth does each flow receive?
- Macroscopic TCP modeling:
  - TCP in steady-state implements a type of resource sharing "Max-Min Fairness"
  - Bandwidth allocation can be solved efficiently with appropriate data structure
  - Validated with NS-2 simulators
  - Justified for "long-enough" transfers...

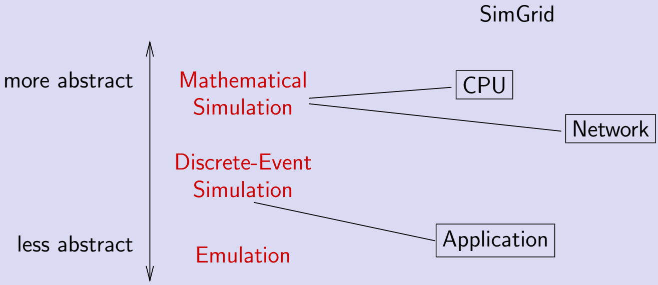
Resource Sharing in SimGrid

- Resource sharing for CPU:
  - process/threads competing for resource get a fair share of the CPU "in steady state"
  - no need to emulate CPU
  - compute the CPU cycles allocated of each process/thread (rate)
- Resource sharing for the network:
  - many end-points, routers and links,
  - many end-to-end TCP flows ?
  - macroscopic behavior: How much bandwidth does each flow receive?
- Macroscopic TCP modeling:
  - TCP in steady-state implements a type of resource sharing "Max-Min Fairness"
  - Bandwidth allocation can be solved efficiently with appropriate data structure
  - Validated with NS-2 simulators
  - Justified for "long-enough" transfers. . .

Resource Sharing in SimGrid

- Resource sharing for CPU:
    - process/threads competing for resource get a fair share of the CPU "in steady state"

SimGrid in a Nutshell



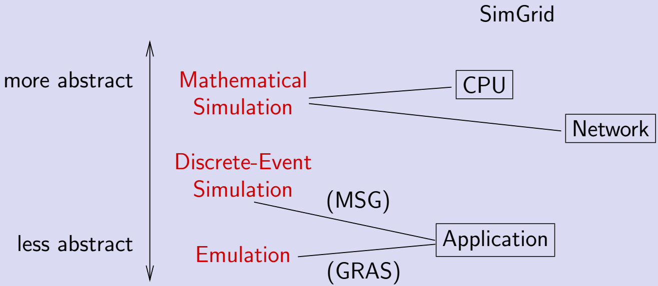
data structure

- Validated with NS-2 simulators
- Justified for "long-enough" transfers. . .

Resource Sharing in SimGrid

- Resource sharing for CPU:
  - process/threads competing for resource get a fair share of the CPU "in steady state"

SimGrid in a Nutshell



SimGrid

more abstract

less abstract

Mathematical Simulation

CPU

Network

Discrete-Event Simulation

(MSG)

Application

Emulation

(GRAS)

data structure

  - Validated with NS-2 simulators
  - Justified for "long-enough" transfers. . .

Simple example: master-slave tasking

Code for slave

```
int slave(int argc, char *argv[]) {
   while(1) {
      m_task_t task = MSG_task_get(&(task), TASK_PORT);
MSG_task_execute(task);    }
}
```

Code for master

```
int master(int argc, char *argv[]) {
   for (i = 0; i < number_of_tasks; i++) {
      tasks[i] = MSG_task_create("task", task_computation_size,
                    task_communication_size, NULL);
   }
   /* [...]  */
   for (i = 0; i < number_of_tasks; i++) {
      m_host_t target_slave = choose_target();
      MSG_task_put(tasks[i], target_slave, TASK_PORT);
   }
}
```

Simple example: master-slave tasking

Platform description

XML file describing:

- CPUs
- network links
- routes (between CPUs, using network links)

Deployment

XML file describing the application:

- which process is run on which host, with argument list

- All simulated processes are run as different threads of a same physical process
- Makes it easy to communicate (shared memory)

A few remarks

SimGrid cannot help you to figure out what is going to be the duration of a real application

but can help you to compare two algorithms

SimGrid cannot model accurately the behavior of a computing platform

but can help you to study the robustness of your algorithm in a noisy environment

SimGrid cannot help you to fix some experimental thresholds

but can be used to design adaptive thresholds strategies and test them against a wide variety of environments

SimGrid cannot help you to debug an already existing code

but can help you to test and debug your algorithms before the real implementation

A few remarks

SimGrid cannot help you to figure out what is going to be the duration of a real application
but can help you to compare two algorithms

SimGrid cannot model accurately the behavior of a computing platform
but can help you to study the robustness of your algorithm in a noisy environment

SimGrid cannot help you to fix some experimental thresholds
but can be used to design adaptive thresholds strategies and test them against a wide variety of environments

SimGrid cannot help you to debug an already existing code
but can help you to test and debug your algorithms before the real implementation

A few remarks

SimGrid cannot help you to figure out what is going to be the duration of a real application

but can help you to compare two algorithms

SimGrid cannot model accurately the behavior of a computing platform

but can help you to study the robustness of your algorithm in a noisy environment

SimGrid cannot help you to fix some experimental thresholds

but can be used to design adaptive thresholds strategies and test them against a wide variety of environments

SimGrid cannot help you to debug an already existing code

but can help you to test and debug your algorithms before the real implementation

A few remarks

SimGrid cannot help you to figure out what is going to be the duration of a real application

but can help you to compare two algorithms

SimGrid cannot model accurately the behavior of a computing platform

but can help you to study the robustness of your algorithm in a noisy environment

SimGrid cannot help you to fix some experimental thresholds

but can be used to design adaptive thresholds strategies and test them against a wide variety of environments

SimGrid cannot help you to debug an already existing code

but can help you to test and debug your algorithms before the real implementation

A few remarks

SimGrid cannot help you to figure out what is going to be the duration of a real application

but can help you to compare two algorithms

SimGrid cannot model accurately the behavior of a computing platform

but can help you to study the robustness of your algorithm in a noisy environment

SimGrid cannot help you to fix some experimental thresholds

but can be used to design adaptive thresholds strategies and test them against a wide variety of environments

SimGrid cannot help you to debug an already existing code

but can help you to test and debug your algorithms before the real implementation

A few remarks

SimGrid cannot help you to figure out what is going to be the duration of a real application

but can help you to compare two algorithms

SimGrid cannot model accurately the behavior of a computing platform

but can help you to study the robustness of your algorithm in a noisy environment

SimGrid cannot help you to fix some experimental thresholds

but can be used to design adaptive thresholds strategies and test them against a wide variety of environments

SimGrid cannot help you to debug an already existing code

but can help you to test and debug your algorithms before the real implementation

A few remarks

SimGrid cannot help you to figure out what is going to be the duration of a real application

but can help you to compare two algorithms

SimGrid cannot model accurately the behavior of a computing platform

but can help you to study the robustness of your algorithm in a noisy environment

SimGrid cannot help you to fix some experimental thresholds

but can be used to design adaptive thresholds strategies and test them against a wide variety of environments

SimGrid cannot help you to debug an already existing code

but can help you to test and debug your algorithms before the real implementation

A few remarks

SimGrid cannot help you to figure out what is going to be the duration of a real application
 but can help you to compare two algorithms

SimGrid cannot model accurately the behavior of a computing platform
 but can help you to study the robustness of your algorithm in a noisy environment

SimGrid cannot help you to fix some experimental thresholds
 but can be used to design adaptive thresholds strategies and test them against a wide variety of environments

SimGrid cannot help you to debug an already existing code
 but can help you to test and debug your algorithms before the real implementation

Bibliography

Henri Casanova.
Simgrid: A toolkit for the simulation of application scheduling.
In *Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'01)*, May 2001.

Henri Casanova, Arnaud Legrand, and Loris Marchal.
Scheduling distributed applications: the SimGrid simulation framework.
In *Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, May 2003.

A. Legrand, M. Quinson, K. Fujiwara, H. Casanova
The SimGrid Project - Simulation and Deployment of Distributed Applications
POSTER in *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC-15)*, Paris, France, May 2006.

http://simgrid.gforge.inria.fr/