Steady-State and Decentralized Scheduling

Loris MARCHAL

(joint work with Olivier BEAUMONT,
Arnaud LEGRAND and Yves ROBERT)

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

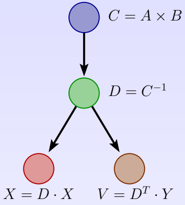
loris.marchal@ens-lyon.fr

EPIT spring school
June 7, 2007

Traditional scheduling

Models:

- Applications:
  task graphs
- Computing platforms:
  homogeneous or heterogeneous processors
  interconnexion network with or without congestion



Objectives:

- allocation $\text{alloc}(T)$: processor computing task T
- schedule $\sigma(T)$: starting time of T.
- optimal makespan

Results:

- NP-hard problem even for independent tasks
- approximation algorithms (list heuristics)

Traditional scheduling

Models:

- Applications:
  task graphs
- Computing platforms:
  homogeneous or heterogeneous processors
  interconnexion network with or without congestion

Objectives:

- allocation $\mathrm{alloc}(T)$: processor computing task T
- schedule $\sigma(T)$: starting time of T.
- optimal makespan

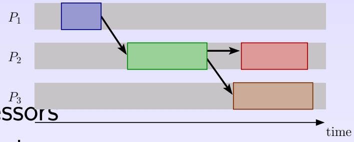
Results:

- NP-hard problem even for independent tasks
- approximation algorithms (list heuristics)

Traditional scheduling

Models:

- Applications:
  task graphs
- Computing platforms:
  homogeneous or heterogeneous processors
  interconnexion network with or without congestion



Objectives:

- allocation $\mathrm{alloc}(T)$: processor computing task T
- schedule $\sigma(T)$: starting time of T.
- optimal makespan

Results:

- NP-hard problem even for independent tasks
- approximation algorithms (list heuristics)

Traditional scheduling

Models:

- Applications:
  task graphs
- Computing platforms:
  homogeneous or heterogeneous processors
  interconnexion network with or without congestion

Objectives:

- allocation $\mathrm{alloc}(T)$: processor computing task T
- schedule $\sigma(T)$: starting time of T.
- optimal makespan

Results:

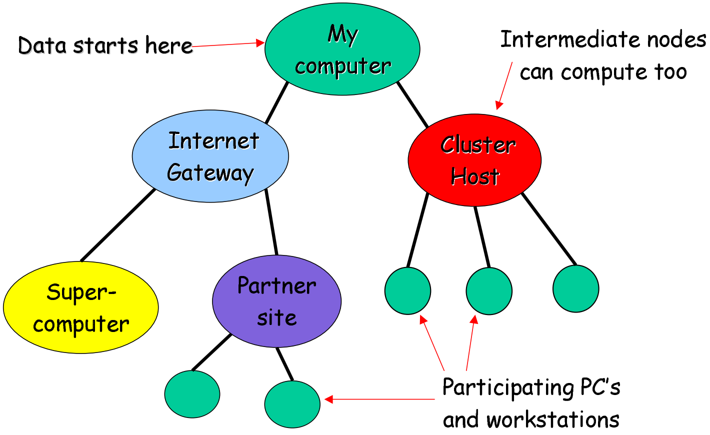
- NP-hard problem even for independent tasks
- approximation algorithms (list heuristics)

Design of efficient schedules

- Approximation algorithm
  example: 3/2-approximation
  optimal: 2 seconds $\rightsquigarrow$ 3 seconds
  optimal: 2 hours $\rightsquigarrow$ 3 hours

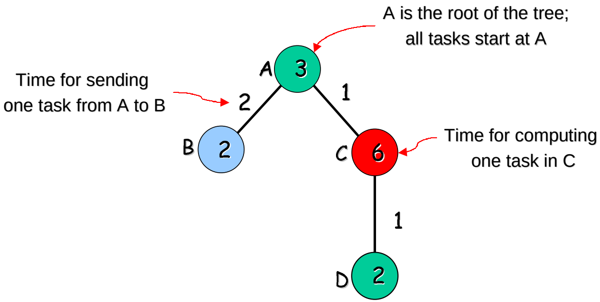
- Asymptotically optimal algorithm
  example: $T_{\text{opt}} + O(1)$
  optimal: 2 seconds $\rightsquigarrow$ 5 minutes $+$ 2 seconds
  optimal: 2 hours $\rightsquigarrow$ 2 hours $+$ 5 minutes

Example

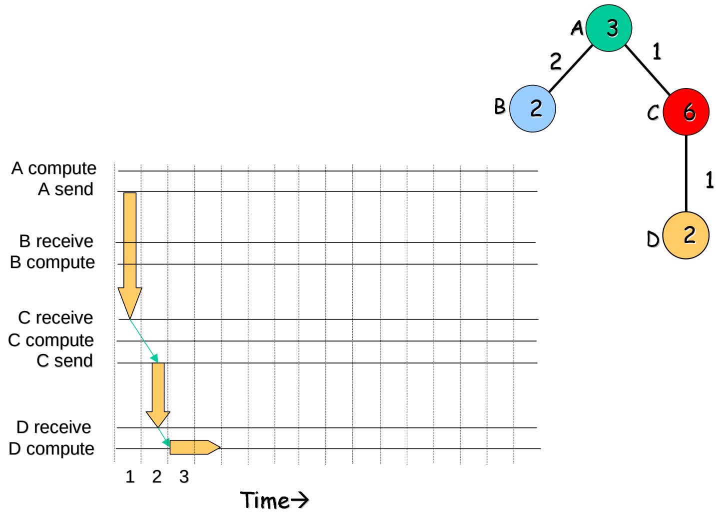


Data starts here
My computer
Intermediate nodes can compute too
Internet Gateway
Cluster Host
Super-computer
Partner site
Participating PC's and workstations

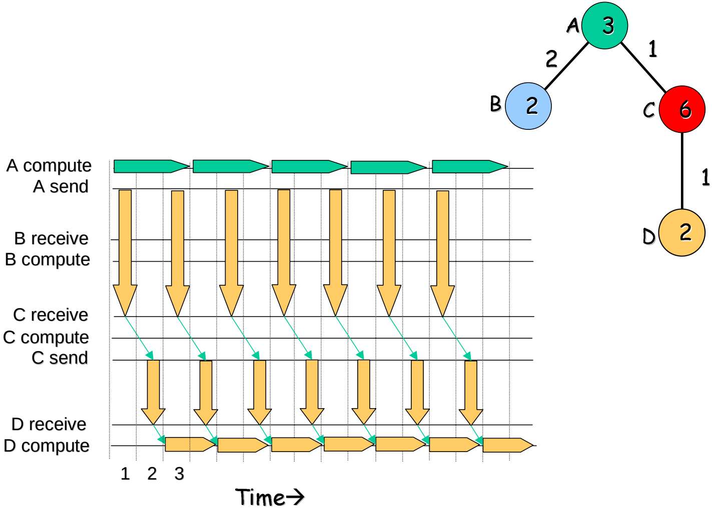
Example


A is the root of the tree;
all tasks start at A
Time for sending
one task from A to B
Time for computing
one task in C
A
3
B
2
C
6
D
2
2
1
1

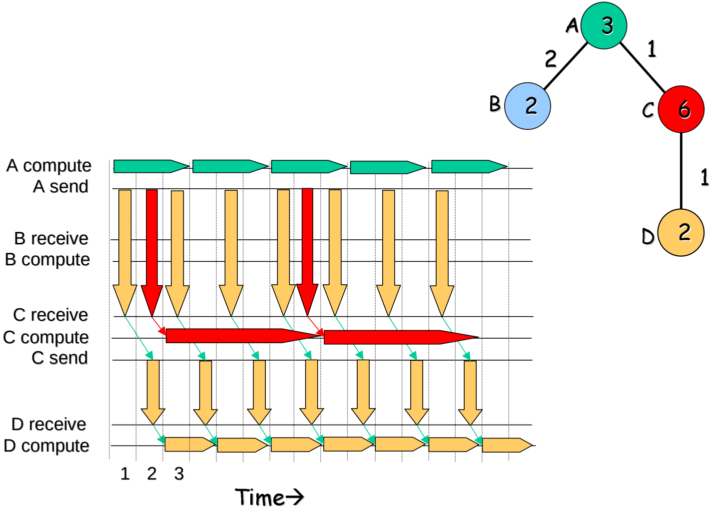
Example


A
3
2
1
B
2
C
6
1
D
2
A compute
A send
B receive
B compute
C receive
C compute
C send
D receive
D compute
1
2
3
Time→

Example

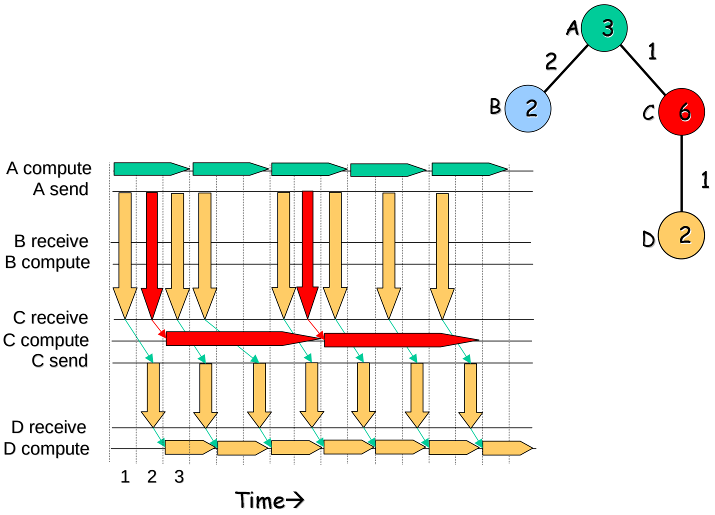

A
3
2
1
B
2
C
6
1
D
2
A compute
A send
B receive
B compute
C receive
C compute
C send
D receive
D compute
1
2
3
Time→

Example



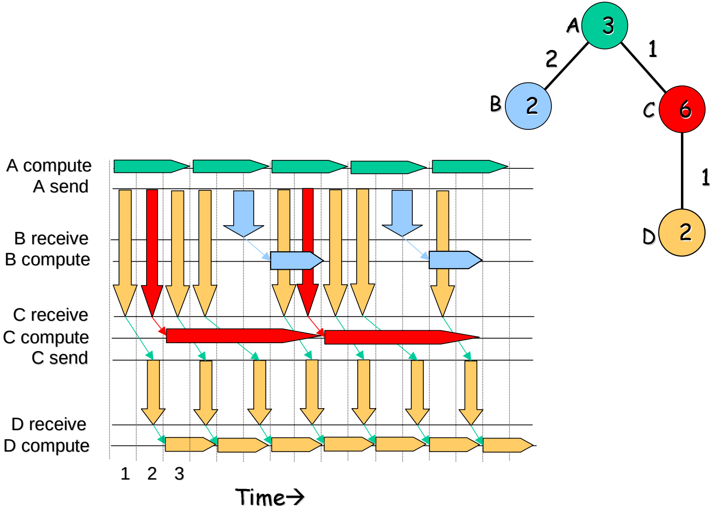
A
3
2
1
B
2
C
6
1
D
2
A compute
A send
B receive
B compute
C receive
C compute
C send
D receive
D compute
1
2
3
Time→

Example

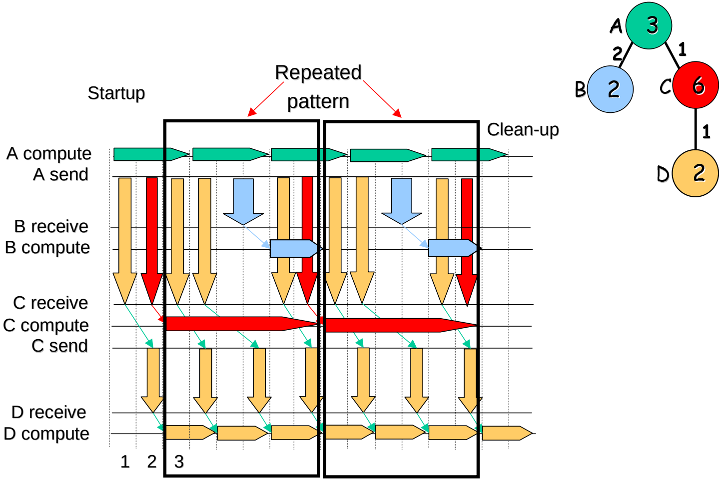


A compute
A send
B receive
B compute
C receive
C compute
C send
D receive
D compute
1
2
3
Time→
A
3
2
1
B
2
C
6
1
D
2

Example


A
3
2
1
B
2
C
6
1
D
2
A compute
A send
B receive
B compute
C receive
C compute
C send
D receive
D compute
1 2 3
Time→

Example



Steady-state: 7 tasks every 6 time units

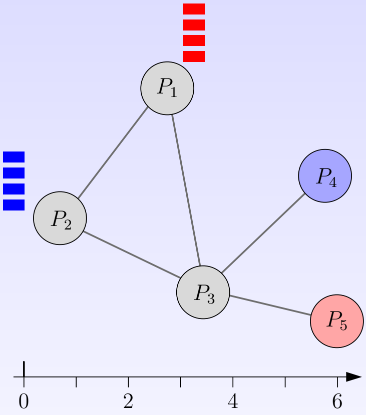
Outline

Steady-State Scheduling
- Packet routing
- Problem formulation
- Problem solving in the general case
- Simplification in the bidirectional case
- Moving to general task graphs
- Collective communications

Towards distributed scheduling
- Limits of static steady-state scheduling
- Dynamic scheduling for independent tasks

Outline

Packet routing without fixed routing

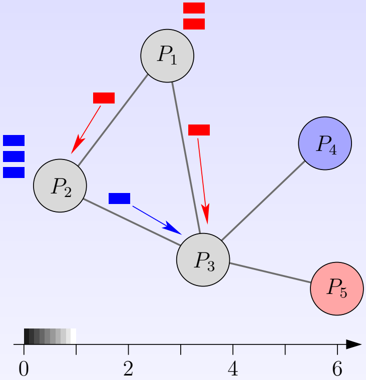


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: 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 routing

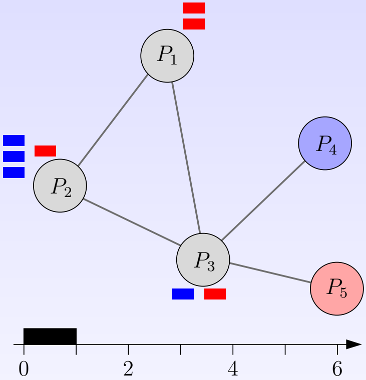


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: 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 routing

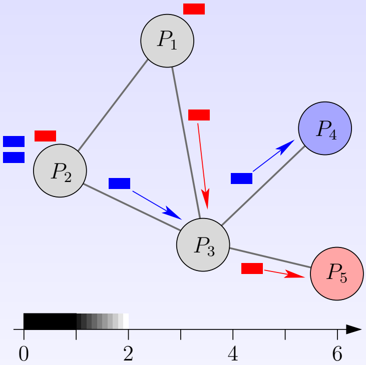


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: 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 routing

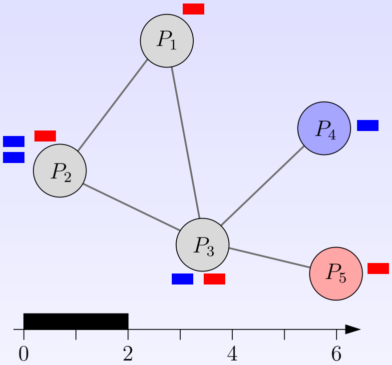


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: 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 routing

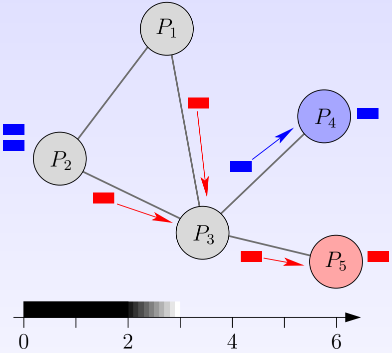


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n^{k,l}_{i,j}$: 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 routing

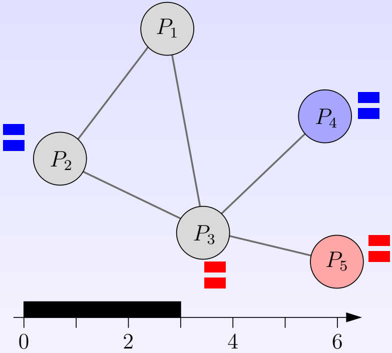


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

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

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

Packet routing without fixed routing

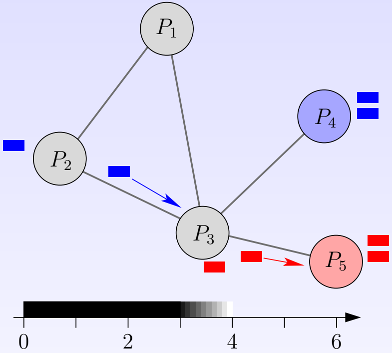


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

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

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

Packet routing without fixed routing

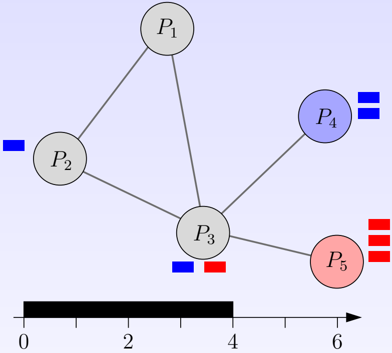


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: 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 routing

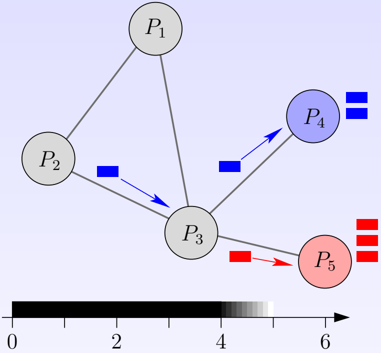


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: 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 routing

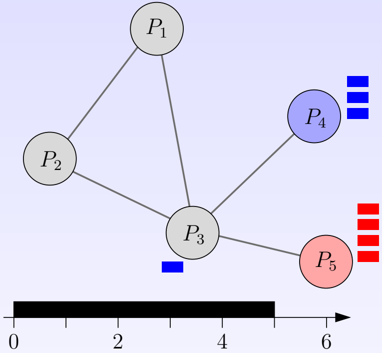


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: 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 routing

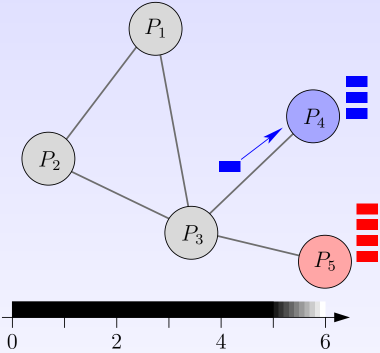


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n^{k,l}_{i,j}$: 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 routing

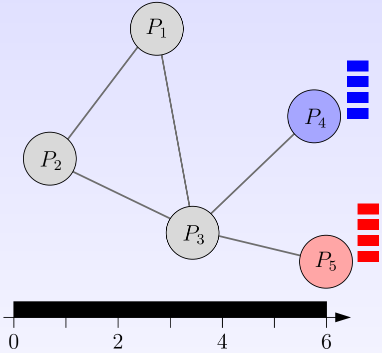


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: 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 routing

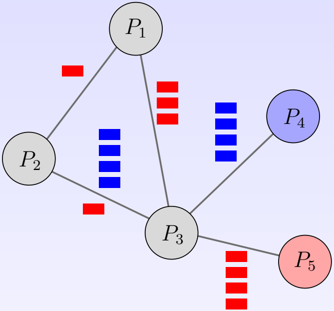


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: 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 routing

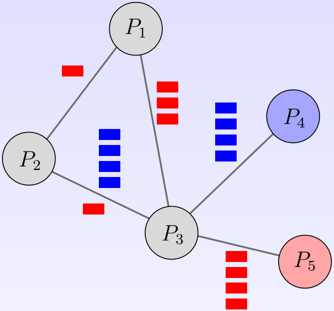


- n_c collections of packets to be routed
- packets of a same collection may follow different paths
- $n^{k,l}$: total number of packets to be routed from k to l
- rule: one edge cannot carry two packets at the same time

- $n_{i,j}^{k,l}$: 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 routing



- 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:

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

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

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

$$\text{Minimize } C_{\max}$$

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

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

$$\text{Minimize } C_{\max}$$

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

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

$$\text{Minimize } C_{\max}$$

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:

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

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

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

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:
$$C_{\max} \le C^* \le C_{\max} + \sqrt{C_{\max} n_c} + 2|A|\sqrt{C_{\max} n_c}|V| + |A| n_c |V|$$
$$C^* = C_{\max} + O(\sqrt{C_{\max}})$$

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
- 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

$\Rightarrow$ 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

$\Rightarrow$ 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

$\Rightarrow$ 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

$\Rightarrow$ 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

$\Rightarrow$ react on the fly to resource availability variations

Outline

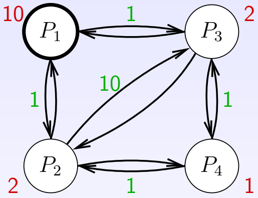
Platform model

Parameters

- computing speed: w_i
  computation time of n tasks: $n \times w_i$
- communication speed $c_{j,k}$
  communication time of n tasks: $n \times c_{j,k}$.

Interactions

- full communication/computation overlap
- Bidirectional 1-port model:
  - while P_j sends a message to P_k
  - P_j cannot send other messages
  - P_k cannot receive other messages
- Unidirectional 1-port model:
  - while P_j sends a message to P_k
  - P_j and P_k cannot send or receive



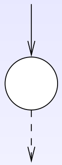
Application model

Steady-state scheduling applies to different problems:

- independent tasks,
- task graphs (DAGs)
- communications (broadcast. . .)

steady-state version of these problems:

- series (large collection) of independent tasks
- series of identical DAGs
- series of broadcasts



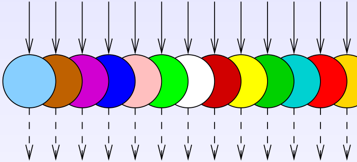
Application model

Steady-state scheduling applies to different problems:

- independent tasks,
- task graphs (DAGs)
- communications (broadcast. . .)

steady-state version of these problems:

- series (large collection) of independent tasks
- series of identical DAGs
- series of broadcasts

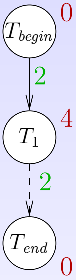


Allocations and independent sets

Problem to solve Execute a series of DAGs on this platform. Optimize the throughput.

Allocation An allocation describes where is executed one DAG of this series: pair of mapping π : {nodes of the DAG} $\longrightarrow$ {nodes of the platform} and σ : {edges of the DAG} $\longrightarrow$ {paths of the platform}

Independent pattern Set of operations (communication and computation) which can be done simultaneously according to the model.



(N copies)

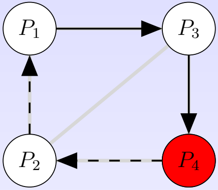
Allocations and independent sets

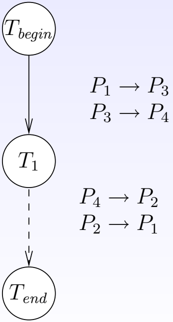
Problem to solve Execute a series of DAGs on this platform. Optimize the throughput.

Allocation An allocation describes where is executed one DAG of this series: pair of mapping π : {nodes of the DAG} $\longrightarrow$ {nodes of the platform}
and σ : {edges of the DAG} $\longrightarrow$ {paths of the platform}

Independent pattern Set of operations (communication and computation) which can be done simultaneously according to the model.

examples:





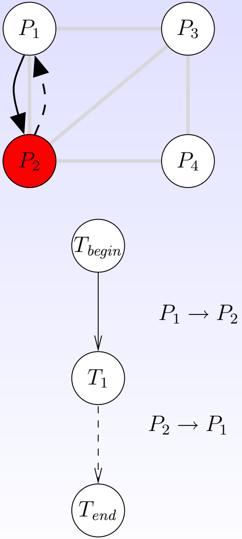
Allocations and independent sets

Problem to solve Execute a series of DAGs on this platform. Optimize the throughput.

Allocation An allocation describes where is executed one DAG of this series: pair of mapping $\pi : \{\text{nodes of the DAG}\} \longrightarrow \{\text{nodes of the platform}\}$ and $\sigma : \{\text{edges of the DAG}\} \longrightarrow \{\text{paths of the platform}\}$

Independent pattern Set of operations (communication and computation) which can be done simultaneously according to the model.

examples:



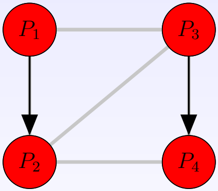
Allocations and independent sets

Problem to solve Execute a series of DAGs on this platform. Optimize the throughput.

Allocation An allocation describes where is executed one DAG of this series: pair of mapping π : {nodes of the DAG} $\longrightarrow$ {nodes of the platform}
and σ : {edges of the DAG} $\longrightarrow$ {paths of the platform}

Independent pattern Set of operations (communication and computation) which can be done simultaneously according to the model.

examples under the unidirectional one-port model:



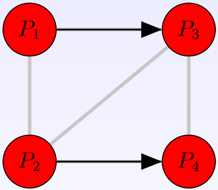
Allocations and independent sets

Problem to solve Execute a series of DAGs on this platform. Optimize the throughput.

Allocation An allocation describes where is executed one DAG of this series: pair of mapping π : {nodes of the DAG} $\longrightarrow$ {nodes of the platform} and σ : {edges of the DAG} $\longrightarrow$ {paths of the platform}

Independent pattern Set of operations (communication and computation) which can be done simultaneously according to the model.

examples under the unidirectional one-port model:



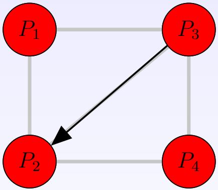
Allocations and independent sets

Problem to solve Execute a series of DAGs on this platform. Optimize the throughput.

Allocation An allocation describes where is executed one DAG of this series: pair of mapping π : {nodes of the DAG} $\longrightarrow$ {nodes of the platform} and σ : {edges of the DAG} $\longrightarrow$ {paths of the platform}

Independent pattern Set of operations (communication and computation) which can be done simultaneously according to the model.

examples under the unidirectional one-port model:



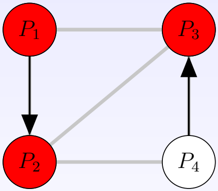
Allocations and independent sets

Problem to solve Execute a series of DAGs on this platform. Optimize the throughput.

Allocation An allocation describes where is executed one DAG of this series: pair of mapping π : $\{$nodes of the DAG$\} \longrightarrow \{$nodes of the platform$\}$ and σ : $\{$edges of the DAG$\} \longrightarrow \{$paths of the platform$\}$

Independent pattern Set of operations (communication and computation) which can be done simultaneously according to the model.

examples under the unidirectional one-port model:



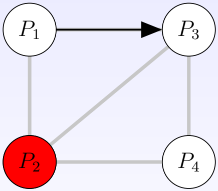
Allocations and independent sets

Problem to solve Execute a series of DAGs on this platform. Optimize the throughput.

Allocation An allocation describes where is executed one DAG of this series: pair of mapping π : {nodes of the DAG} $\longrightarrow$ {nodes of the platform}
and σ : {edges of the DAG} $\longrightarrow$ {paths of the platform}

Independent pattern Set of operations (communication and computation) which can be done simultaneously according to the model.

examples under the unidirectional one-port model:



Defining a schedule

A schedule is described by:

- a set of allocations A_a with weights α_a
  "in a period, allocation A_a is used during α_a seconds"
- a set of independent patterns P_p with weights β_s
  "in a period, pattern P_p is used during β_p seconds"

Definition of the throughput: $\sum_a \alpha_a$

Building a schedule

- a set of allocations A_a with weights α_a
- a set of independent patterns P_p with weights β_p

with some conditions:

- total time for independent patterns is at most period length:

$$\sum_{p} \beta_p \leq 1$$

- each resource utilization time in the allocation is less than resource availability in the independent patterns:

$$\forall r, \qquad \sum_{A_a \ni r} \alpha_a \leq \sum_{P_p \ni r} \beta_p$$

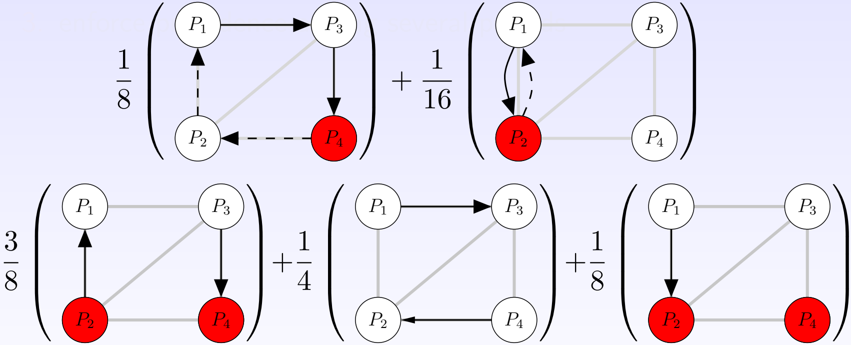
Theorem.

These conditions are sufficient to build a periodic schedule.

Building a schedule – example

input: allocation and patterns

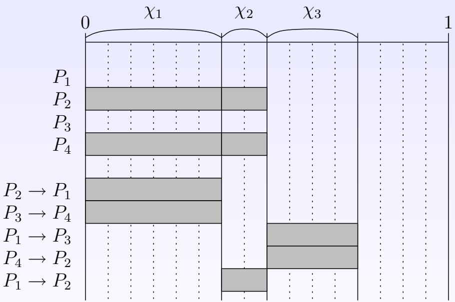
1. build allocation spots in patterns: one period

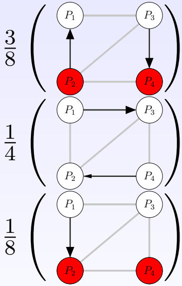


Building a schedule – example

input: allocation and patterns

1. build allocation spots in patterns: one period
2. avoid fractional number of messages
3. enforce precedence among several periods

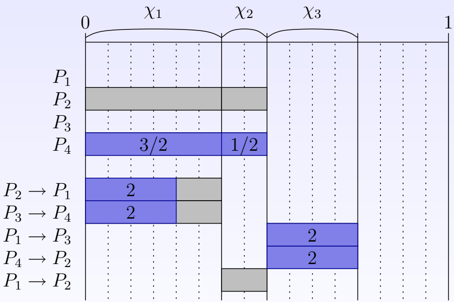


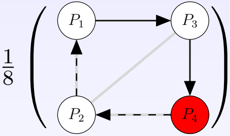


Building a schedule – example

input: allocation and patterns

1. build allocation spots in patterns: one period
2. avoid fractional number of messages ($\times$ PPCM)
3. enforce precedence among several periods

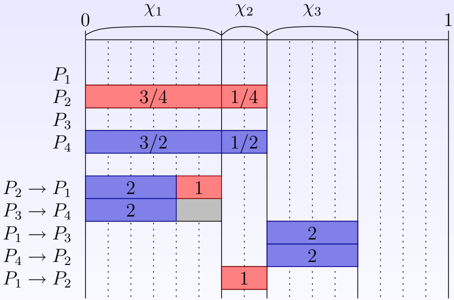


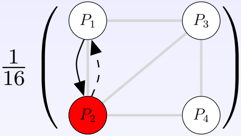


Building a schedule – example

input: allocation and patterns

1. build allocation spots in patterns: one period
2. avoid fractional number of messages ($\times$ PPCM)
3. enforce precedence among several periods

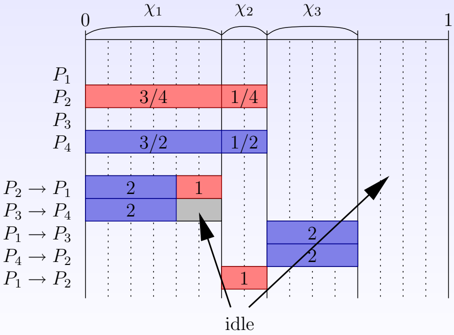




Building a schedule – example

input: allocation and patterns

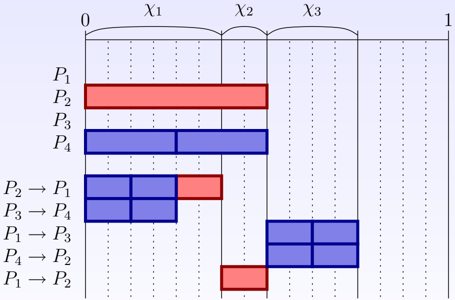
1. build allocation spots in patterns: one period
2. avoid fractional number of messages (× PPCM)
3. enforce precedence among several periods



Building a schedule – example

input: allocation and patterns

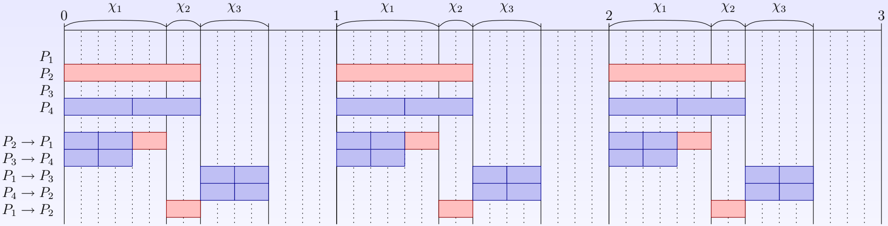
1. build allocation spots in patterns: one period
2. avoid fractional number of messages ($\times$ PPCM)
3. enforce precedence among several periods



Building a schedule – example

input: allocation and patterns

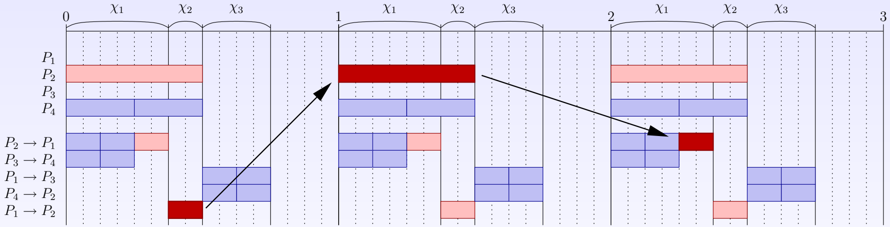
1. build allocation spots in patterns: one period
2. avoid fractional number of messages ($\times$ PPCM)
3. enforce precedence among several periods



Building a schedule – example

input: allocation and patterns

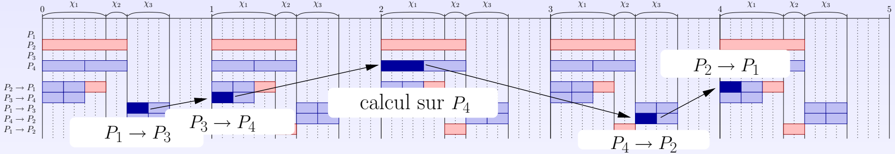
1. build allocation spots in patterns: one period
2. avoid fractional number of messages (× PPCM)
3. enforce precedence among several periods



Building a schedule – example

input: allocation and patterns

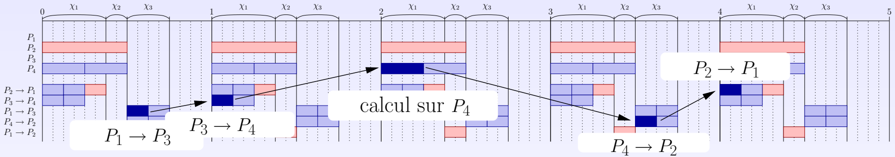
1. build allocation spots in patterns: one period
2. avoid fractional number of messages ($\times$ PPCM)
3. enforce precedence among several periods



Building a schedule – example

input: allocation and patterns

1. build allocation spots in patterns: one period
2. avoid fractional number of messages ($\times$ PPCM)
3. enforce precedence among several periods



This construction is always possible.

General formulation

Find α_a, β_p

Maximize $\rho = \sum_a \alpha_a$

$$\begin{cases} & \sum_p \beta_p \leq 1 \\ \forall r, & \sum_{A_a \ni r} \alpha_a \leq \sum_{P_p \ni r} \beta_p \\ & \alpha_a, \beta_p \geq 0 \end{cases}$$

Some limitations...

- there is an exponential number of variables (and constraints)
- a solution is a priori not described in polynomial space: the problem does not belong to NP

General formulation

Find α_a, β_p

Maximize $\rho = \sum_a \alpha_a$

$$\begin{cases} & \sum_p \beta_p \leq 1 \\ \forall r, & \sum_{A_a \ni r} \alpha_a \leq \sum_{P_p \ni r} \beta_p \\ & \alpha_a, \beta_p \geq 0 \end{cases}$$

Some limitations...

- there is an exponential number of variables (and constraints)
- a solution is a priori not described in polynomial space: the problem does not belong to NP

Outline

Existence of a compact solution

Theorem.

For each solution x of the optimization problem, there exists a solution y of same throughput, described in polynomial space.

In particular,

- y has at most n non-zero variables (n=number of non-trivial constraints in the linear program),
- we can restrict ourselves to a problem in NP.

Sketch of proof:

- consider a point P at a vertex of the polyhedron
- P is solution of a sub-system of the constraints matrix, *mostly* composed of trivial constraints $x_i \geq 0$
- p has *a lot* of variables equal to 0

Solving the linear program

$$(\mathcal{P}): \begin{array}{l}\text{Maximize } c^T \cdot X \\ \begin{cases} A \cdot X \leq b \\ X \geq 0 \end{cases}\end{array} \iff (\mathcal{D}): \begin{array}{l}\text{Minimize } b^T \cdot U \\ \begin{cases} A^T \cdot U \geq c \\ U \geq 0 \end{cases}\end{array}$$

optimization $\Longleftarrow$ separation

Theorem.

Given the polyhedron of $(\mathcal{D})$ described by a separation oracle, there exists a polynomial algorithm which

- finds a solution for the primal problem, or
- proves that the problem has no solution.

Solving the linear program

$$(\mathcal{P}) : \begin{array}{l} \text{Maximize } c^T \cdot X \\ \begin{cases} A \cdot X \leq b \\ X \geq 0 \end{cases} \end{array} \iff (\mathcal{D}) : \begin{array}{l} \text{Minimize } b^T \cdot U \\ \begin{cases} A^T \cdot U \geq c \\ U \geq 0 \end{cases} \end{array}$$

optimization $\Longleftarrow$ separation

Theorem.

Given the polyhedron of $(\mathcal{D})$ described by a separation oracle, there exists a polynomial algorithm which

- finds a solution for the primal problem, or
- proves that the problem has no solution.

Solving the linear program

$$(\mathcal{P}): \begin{array}{l} \text{Maximize } c^T \cdot X \\ \begin{cases} A \cdot X \leq b \\ X \geq 0 \end{cases} \end{array} \quad \Longleftrightarrow \quad (\mathcal{D}): \begin{array}{l} \text{Minimize } b^T \cdot U \\ \begin{cases} A^T \cdot U \geq c \\ U \geq 0 \end{cases} \end{array}$$

optimization $\Longleftarrow$ separation

Theorem.

Given the polyhedron of $(\mathcal{D})$ described by a separation oracle, there exists a polynomial algorithm which

- finds a solution for the primal problem, or
- proves that the problem has no solution.

Separation oracle for the dual

Minimize $b^T \cdot U$

$$(\mathcal{D}) : \begin{cases} U \geq 0 \\ \forall \text{ allocation } A_a \quad \sum_{r \in A_a} U_r/\text{speed}(r) \geq U_0 \\ \forall \text{ pattern } P_p \quad \sum_{r \in P_p} U_r \leq U_1 \end{cases}$$

- one variable U_r per resource (processor/link) (+ variable U_0)
- three types of constraints:
  1. positive constraints: easy to check
  2. check only for the allocation with the minimum weight
  3. check only for the pattern with the maximum weight

$$\text{Solve the dual} \iff \begin{cases} \text{find an allocation with minimum weight} \\ \text{find a pattern with maximum weight} \end{cases}$$

Separation oracle for the dual

Minimize $b^T \cdot U$

$$(\mathcal{D}) : \begin{cases} U \geq 0 \\ \forall \text{ allocation } A_a \quad \sum_{r \in A_a} U_r/\text{speed}(r) \geq U_0 \\ \forall \text{ pattern } P_p \quad \sum_{r \in P_p} U_r \leq U_1 \end{cases}$$

- one variable U_r per resource (processor/link) ($+$ variable U_0)
- three types of constraints:
  1. positive constraints: easy to check
  2. check only for the allocation with the minimum weight
  3. check only for the pattern with the maximum weight

Solve the dual $\Longleftrightarrow \begin{cases} \text{find an allocation with minimum weight} \\ \text{find a pattern with maximum weight} \end{cases}$

Separation oracle for the dual

Minimize $b^T \cdot U$

$$(\mathcal{D}) : \begin{cases} U \geq 0 \\ \forall \text{ allocation } A_a \quad \sum_{r \in A_a} U_r/\text{speed}(r) \geq U_0 \\ \forall \text{ pattern } P_p \quad \sum_{r \in P_p} U_r \leq U_1 \end{cases}$$

- one variable U_r per resource (processor/link) (+ variable U_0)
- three types of constraints:
  1. positive constraints: easy to check
  2. check only for the allocation with the minimum weight
  3. check only for the pattern with the maximum weight

$$\text{Solve the dual} \iff \begin{cases} \text{find an allocation with minimum weight} \\ \text{find a pattern with maximum weight} \end{cases}$$

Separation oracle for the dual

Minimize $b^T \cdot U$

$$(\mathcal{D}) : \begin{cases} U \geq 0 \\ \forall \text{ allocation } A_a \quad \sum_{r \in A_a} U_r / \text{speed}(r) \geq U_0 \\ \forall \text{ pattern } P_p \quad \sum_{r \in P_p} U_r \leq U_1 \end{cases}$$

- one variable U_r per resource (processor/link) (+ variable U_0)
- three types of constraints:
  1. positive constraints: easy to check
  2. check only for the allocation with the minimum weight
  3. check only for the pattern with the maximum weight

$$\text{Solve the dual} \iff \begin{cases} \text{find an allocation with minimum weight} \\ \text{find a pattern with maximum weight} \end{cases}$$

Separation oracle for the dual

Minimize $b^T \cdot U$

$$(\mathcal{D}) : \begin{cases} U \geq 0 \\ \forall \text{ allocation } A_a \quad \sum_{r \in A_a} U_r/\text{speed}(r) \geq U_0 \\ \forall \text{ pattern } P_p \quad \sum_{r \in P_p} U_r \leq U_1 \end{cases}$$

- one variable U_r per resource (processor/link) (+ variable U_0)
- three types of constraints:
  1. positive constraints: easy to check
  2. check only for the allocation with the minimum weight
  3. check only for the pattern with the maximum weight

$$\text{Solve the dual} \iff \begin{cases} \text{find an allocation with minimum weight} \\ \text{find a pattern with maximum weight} \end{cases}$$

Separation oracle for the dual

Minimize $b^T \cdot U$

$$(\mathcal{D}): \begin{cases} U \geq 0 \\ \forall \text{ allocation } A_a \quad \sum_{r \in A_a} U_r/\text{speed}(r) \geq U_0 \\ \forall \text{ pattern } P_p \quad \sum_{r \in P_p} U_r \leq U_1 \end{cases}$$

- one variable U_r per resource (processor/link) (+ variable U_0)
- three types of constraints:
  1. positive constraints: easy to check
  2. check only for the allocation with the minimum weight
  3. check only for the pattern with the maximum weight

Solve the dual $\iff \begin{cases} \text{find an allocation with minimum weight} \\ \text{find a pattern with maximum weight} \end{cases}$

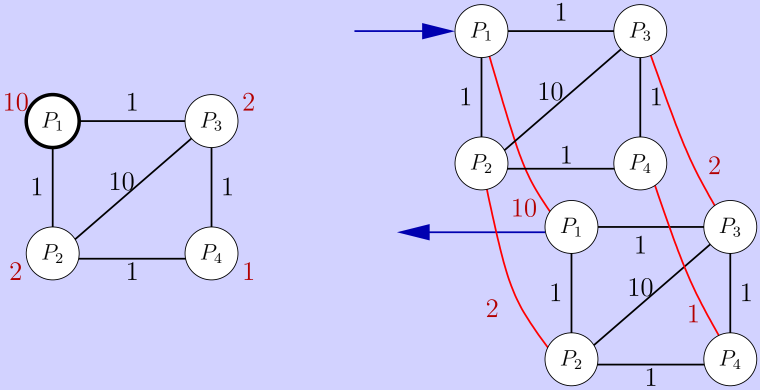
Back to the example: Master-Slave

- Find a minimum-weight allocation:
  $\Leftrightarrow$ find a minimum weight path in a graph obtained from the platform graph
- Find a maximum-weight pattern:
  unidirectional model with overlap:
  ➡ find a maximum-weight matching in the graph
  bidirectional model with overlap:
  ➡ find a maximum-weight in a bipartite graph (duplicate nodes)

$\Rightarrow$ we can solve the Master-Slave problem for both communication models

Back to the example: Master-Slave

- Find a minimum-weight allocation:
  $\Leftrightarrow$ find a minimum weight path in a graph obtained from the platform graph



Back to the example: Master-Slave

- Find a minimum-weight allocation:
  $\Leftrightarrow$ find a minimum weight path in a graph obtained from the platform graph
- Find a maximum-weight pattern:
  unidirectional model with overlap:
  → find a maximum-weight matching in the graph
  bidirectional model with overlap:
  → find a maximum-weight in a bipartite graph (duplicate nodes)

$\Rightarrow$ we can solve the Master-Slave problem for both communication models

Outline

Simplification in the bidirectional case 1/2

Primal linear program:

Maximize $\rho = \sum_a \alpha_a$

$$\begin{cases} & \sum_p \beta_p \leq 1 \\ \forall r, & \sum_{A_a \ni r} \alpha_a \leq \sum_{P_p \ni r} \beta_p \\ & \alpha_a, \beta_p \geq 0 \end{cases}$$

Independent patterns for communications in the bidirectional one-port model: matchings in the bipartite graph constructed from the platform graph

Theorem.

König's theorem for bipartite graphs We can decompose G_B in a weighted sum of matchings of total weight $\leq \delta_{\max}$

Linear program in the bidirectional case

- bound the weighted degree (in and out) of each node in the linear program
- suppress the patterns from the linear program (and use König's theorem to extract them)

Maximise $\rho = \sum_a \alpha_a$

$$\left\{\begin{array}{rll} \forall \text{ CPU } r, & \sum_{A_a \ni r} \alpha_a \leq 1 & \\ \forall \text{ link } r = (i,j), & \sum_{A_a \ni r} \alpha_a \leq T_{i,j} & \\ & \sum_j T_{i,j} \leq 1 & \text{(outgoing communications)} \\ & \sum_i T_{i,j} \leq 1 & \text{(incoming communications)} \\ & \alpha_a, \beta_p \geq 0 & \end{array}\right.$$

We still have a big number of allocations, but. . .

Towards a compact linear program

To go further, we need to specify to an operation:

➡ Back to the Master-Slave example

- Suppress allocations in the linear program
- Use activity variables (close to Bertsimas packet routing)
- Build allocations from the solution of the linear program

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

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

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

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

One-port for incoming communications P_i receives messages sequentially

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

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

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

One-port for incoming communications P_i receives messages sequentially

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

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 incoming 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

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

UNDER THE CONSTRAINTS

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

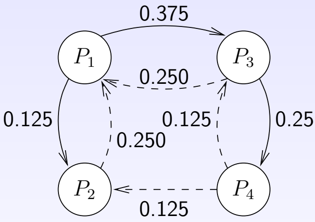
How to extract allocations?

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

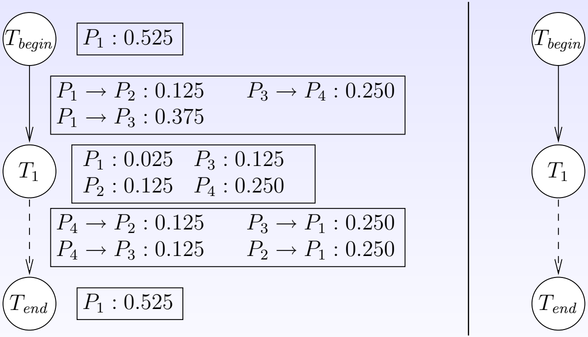
Communications



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

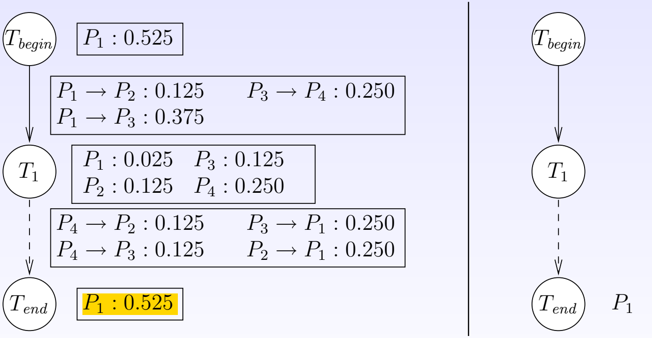
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



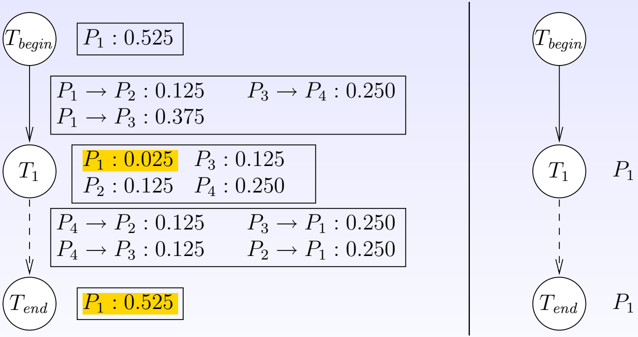
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



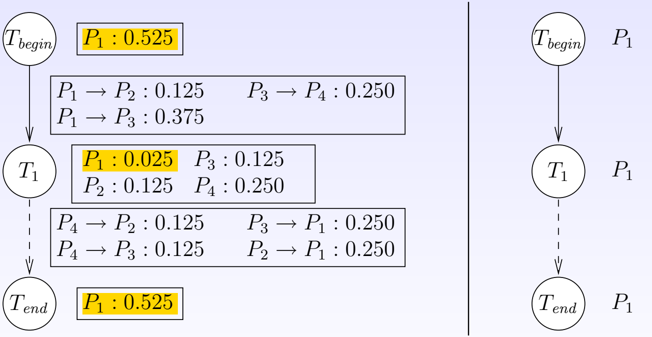
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



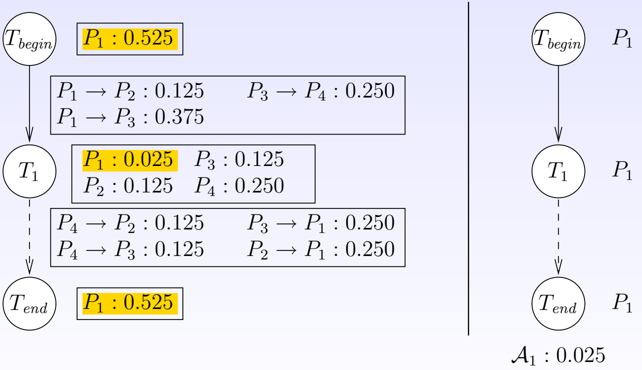
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



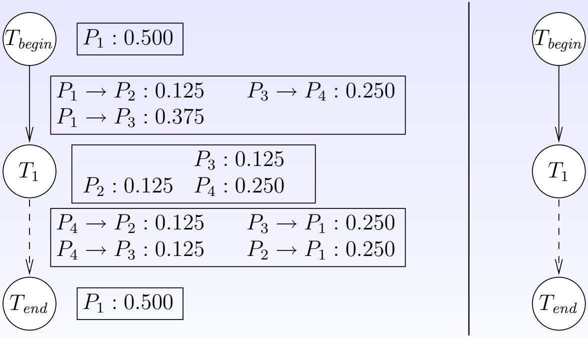
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



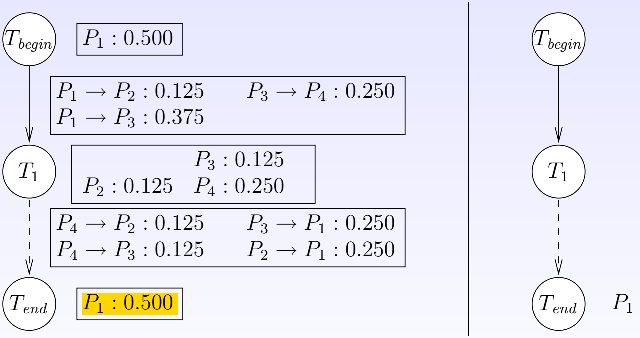
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



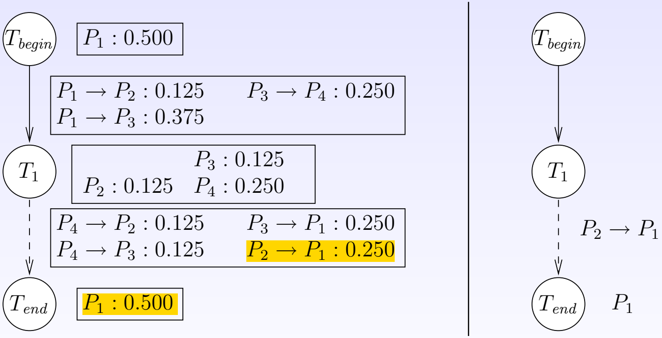
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



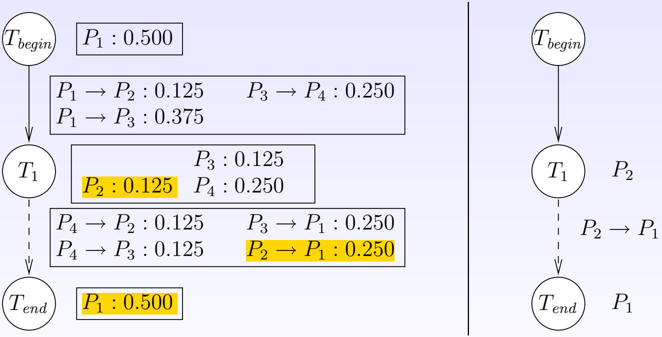
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



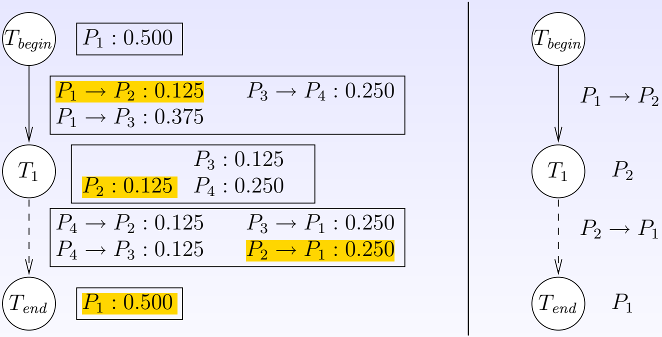
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



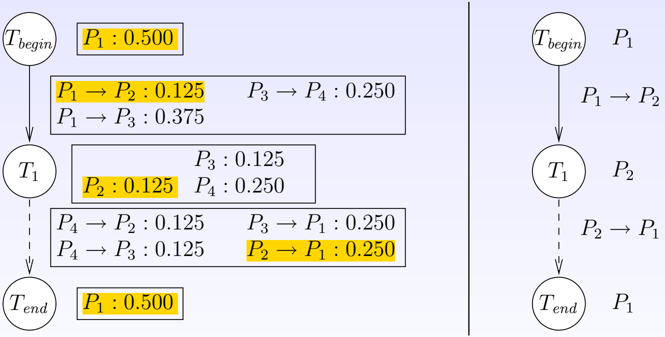
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



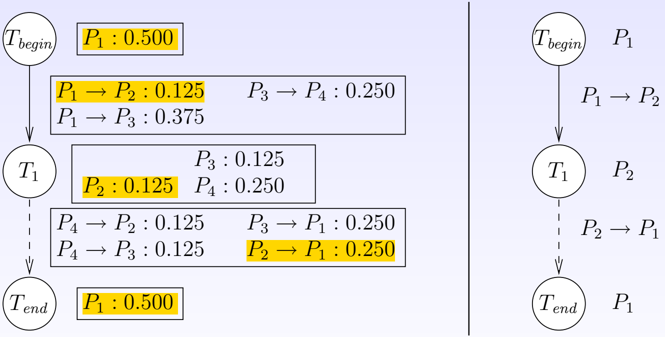
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



Decomposition into a set of allocations (1/2)

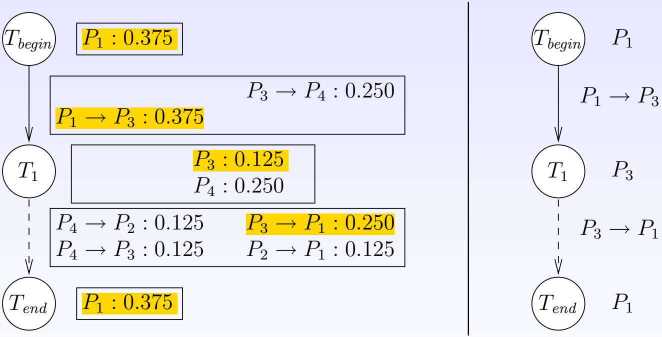
Steady state = superposition of several allocations



$\mathcal{A}_2 : 0.125$

Decomposition into a set of allocations (1/2)

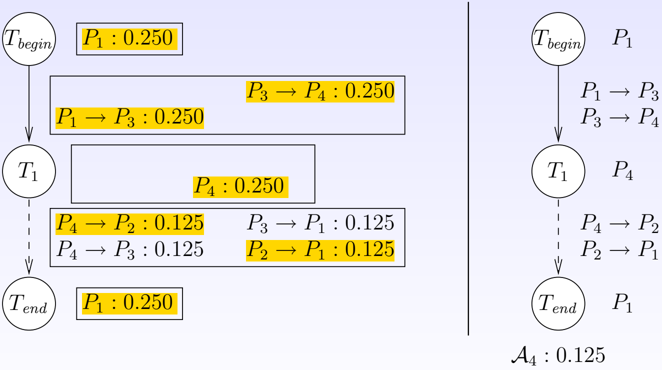
Steady state = superposition of several allocations



$\mathcal{A}_3 : 0.125$

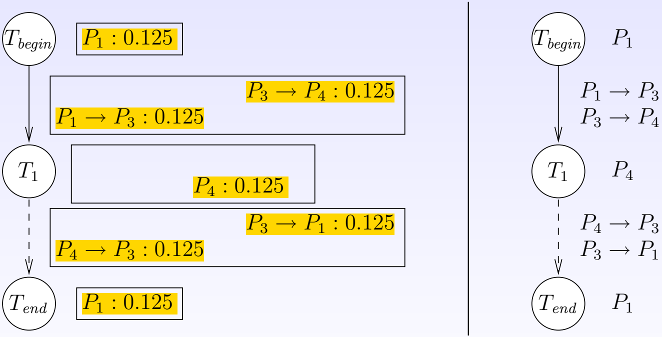
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations



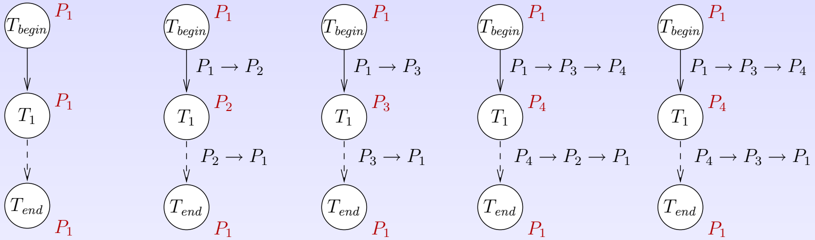
Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations

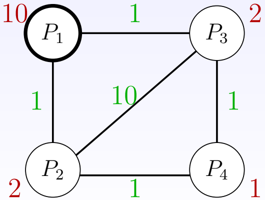


$\mathcal{A}_5 : 0.125$

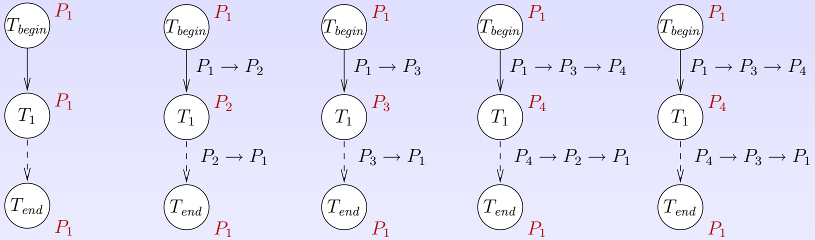
Decomposition into a set of allocations (2/2)



$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$	$\mathcal{A}_5$
0,025	0,125	0,125	0,125	0,125

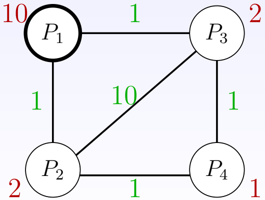


Decomposition into a set of allocations (2/2)

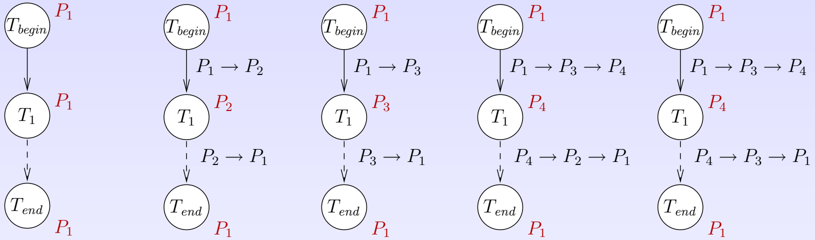


$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$	$\mathcal{A}_5$
0,025	0,125	0,125	0,125	0,125

This decomposition is always possible

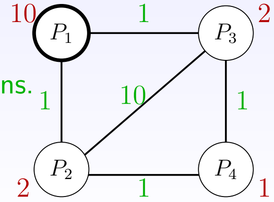


Decomposition into a set of allocations (2/2)

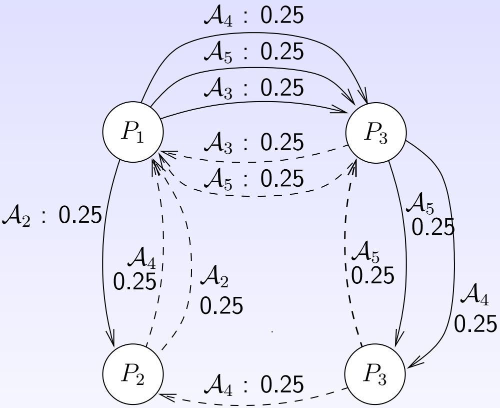


$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$	$\mathcal{A}_5$
0,025	0,125	0,125	0,125	0,125

Then we need patterns to orchestrate communications.

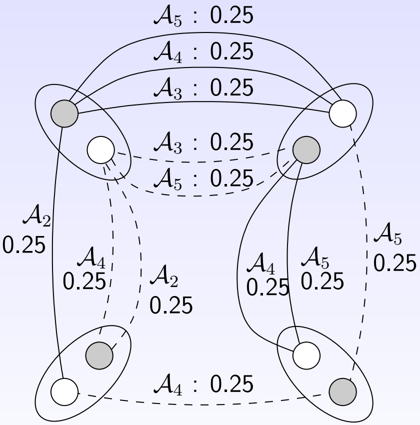


Communication graph

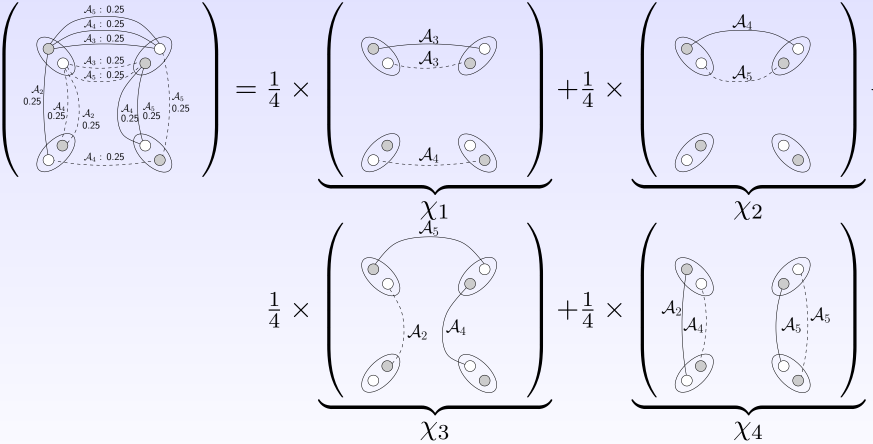


Fraction of time spent transferring some $e_{k,l}$ file from P_i to P_j for a given allocation

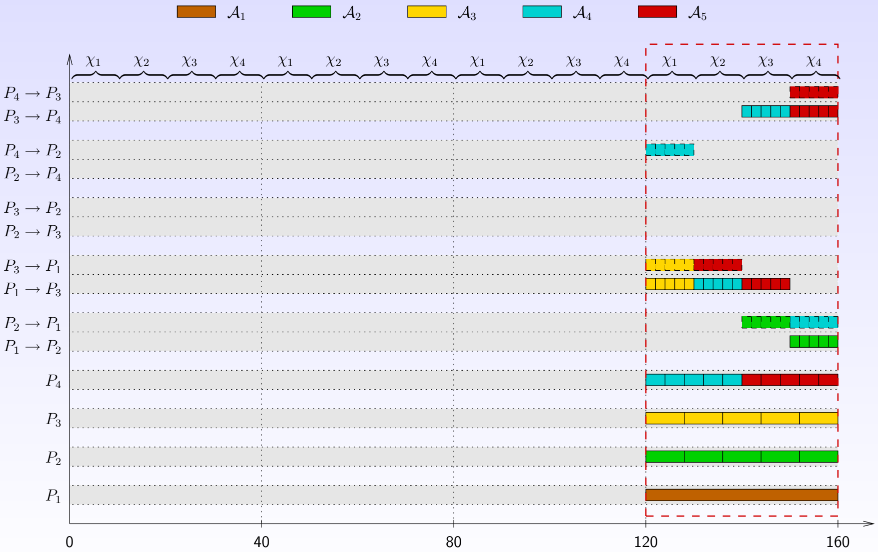
One-port constraints = matching



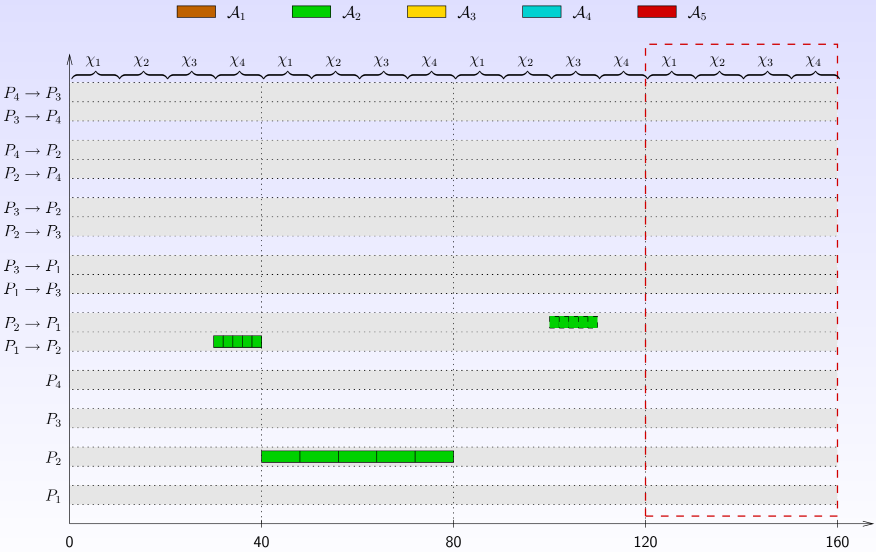
Decomposition into matchings (edge coloring)



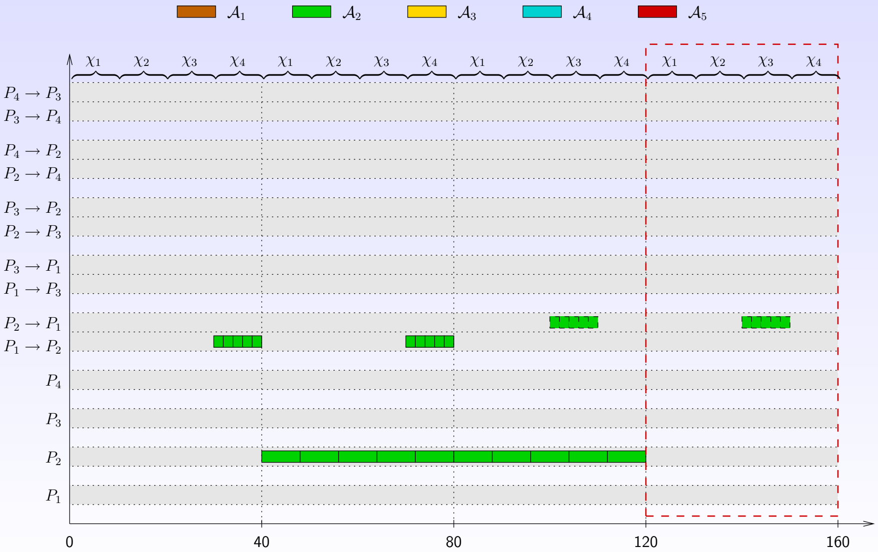
Cyclic scheduling achieving optimal throughput



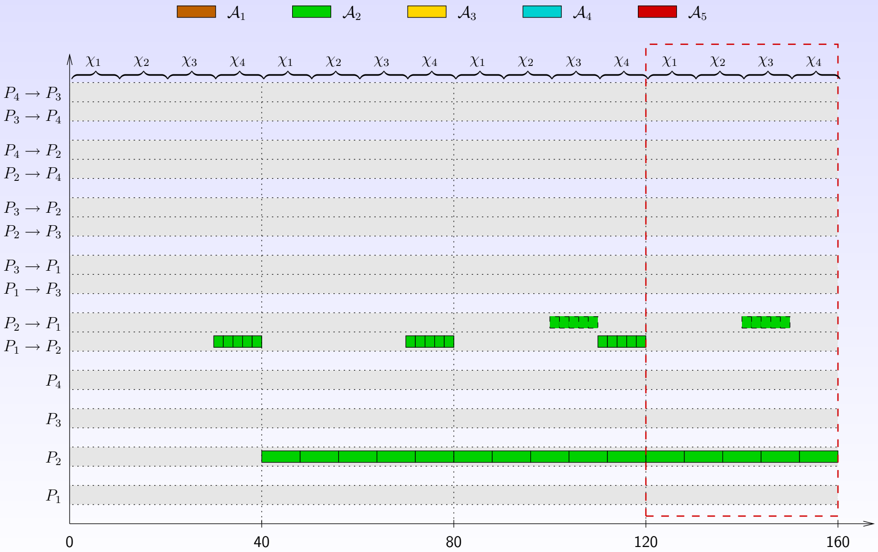
Cyclic scheduling achieving optimal throughput



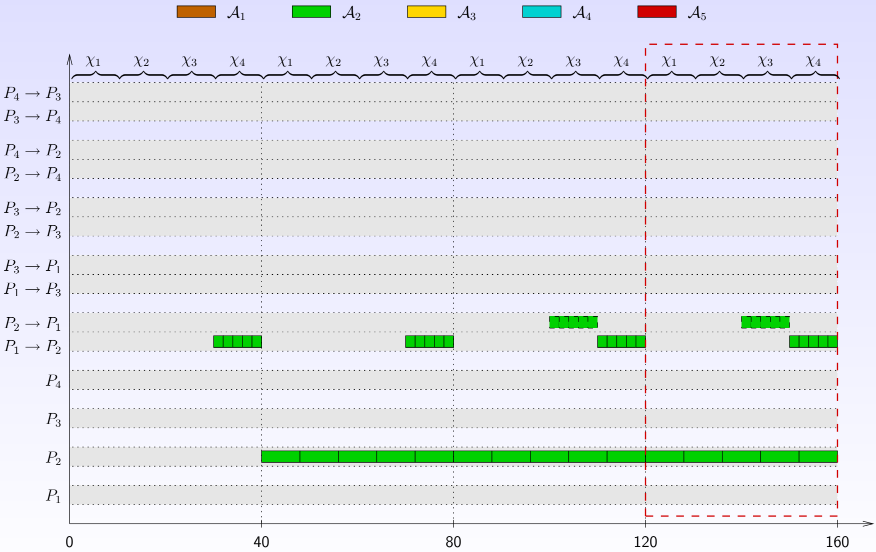
Cyclic scheduling achieving optimal throughput



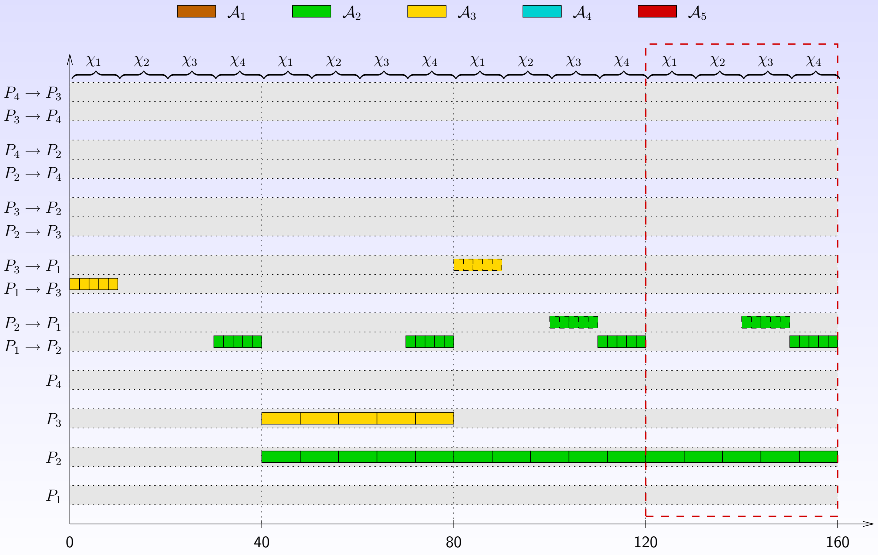
Cyclic scheduling achieving optimal throughput



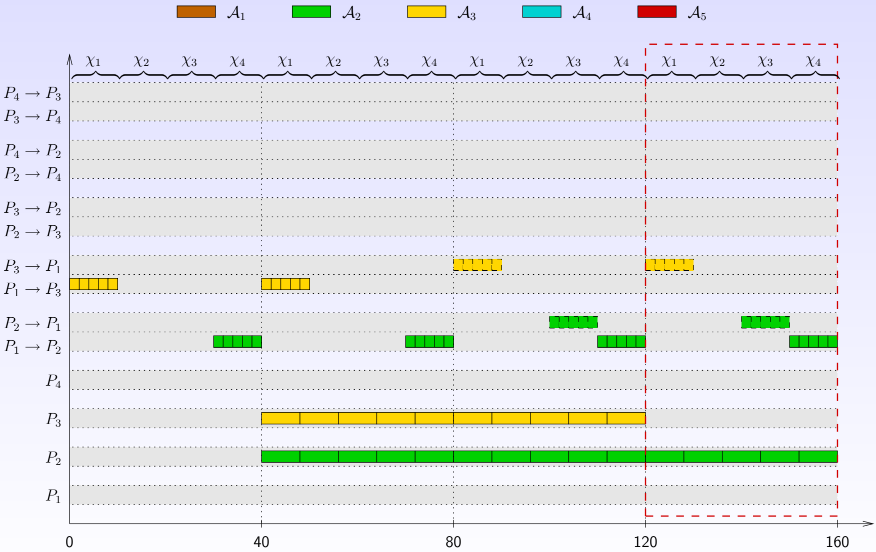
Cyclic scheduling achieving optimal throughput



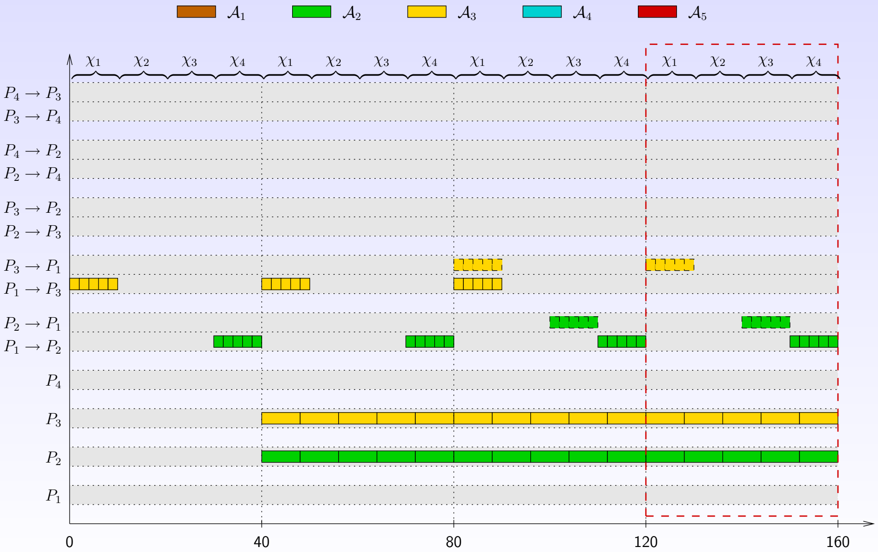
Cyclic scheduling achieving optimal throughput



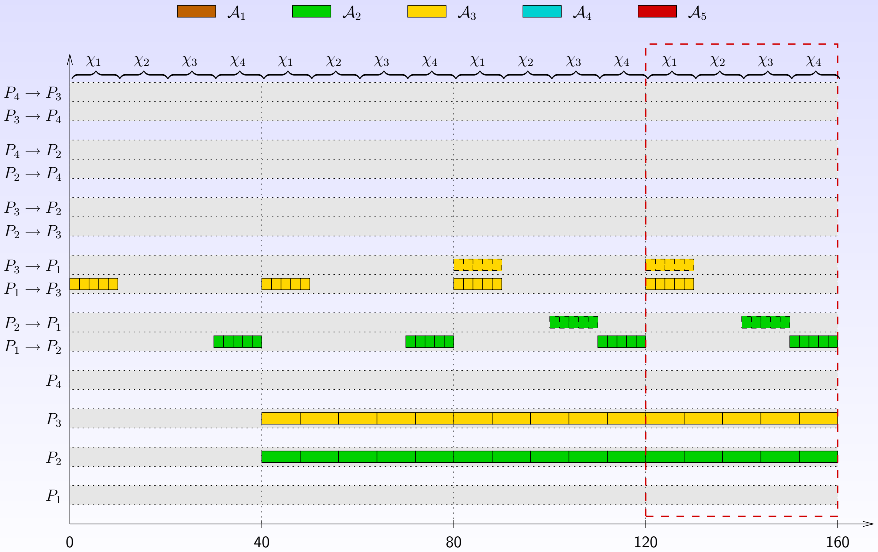
Cyclic scheduling achieving optimal throughput



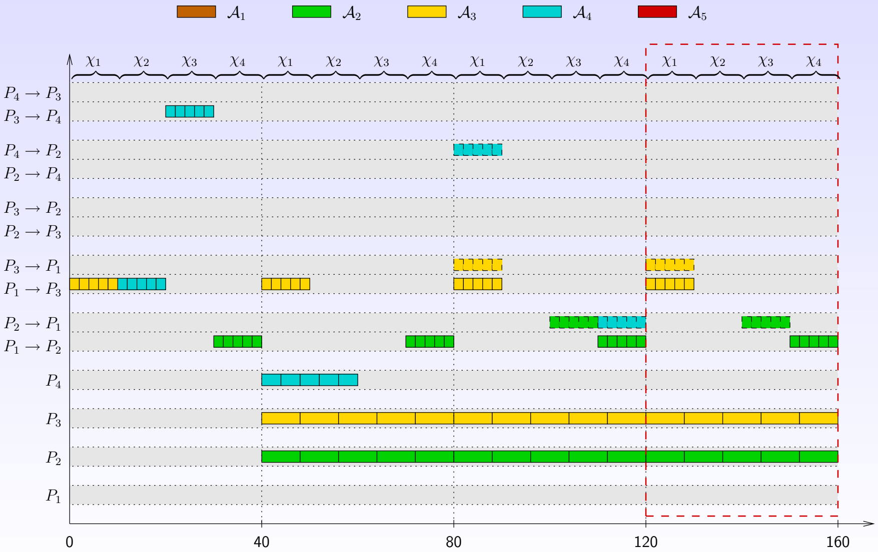
Cyclic scheduling achieving optimal throughput



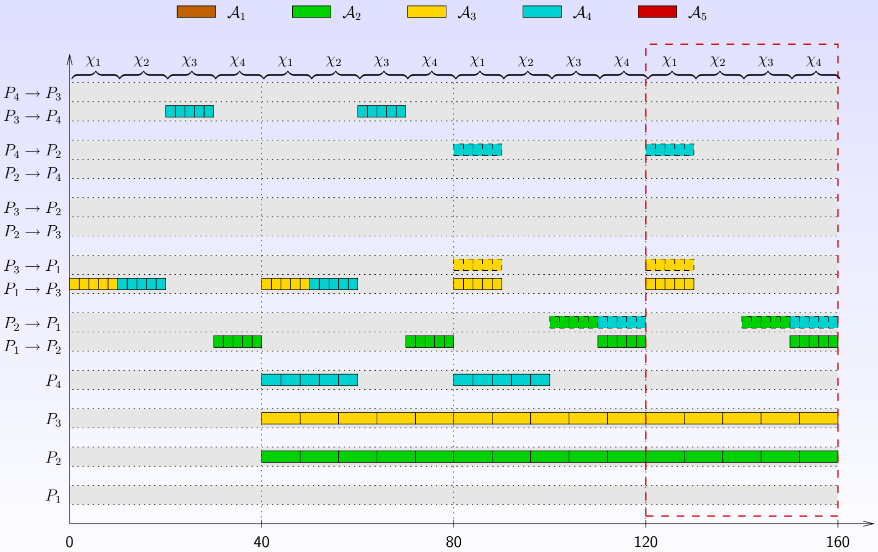
Cyclic scheduling achieving optimal throughput



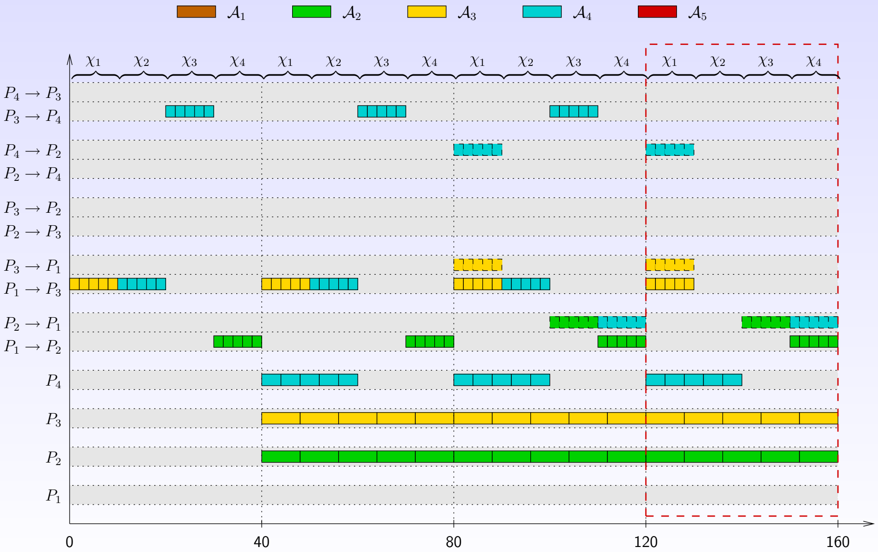
Cyclic scheduling achieving optimal throughput



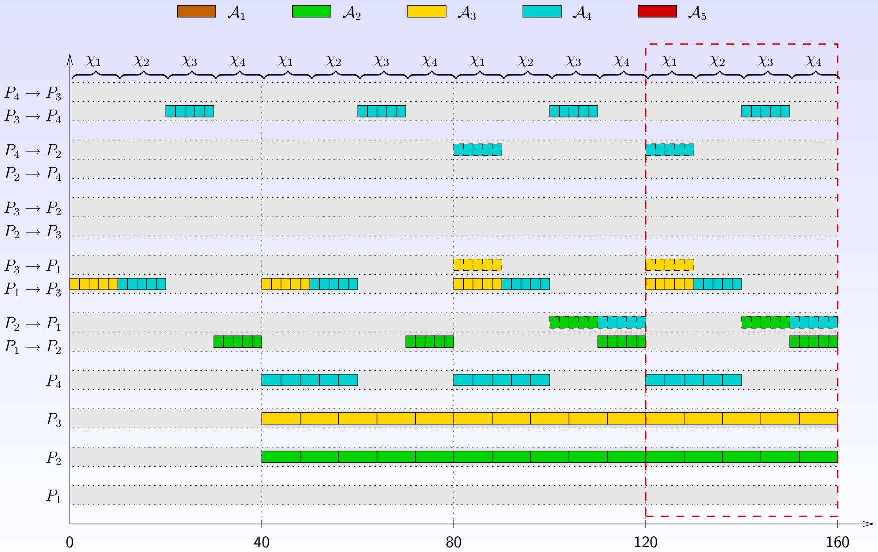
Cyclic scheduling achieving optimal throughput



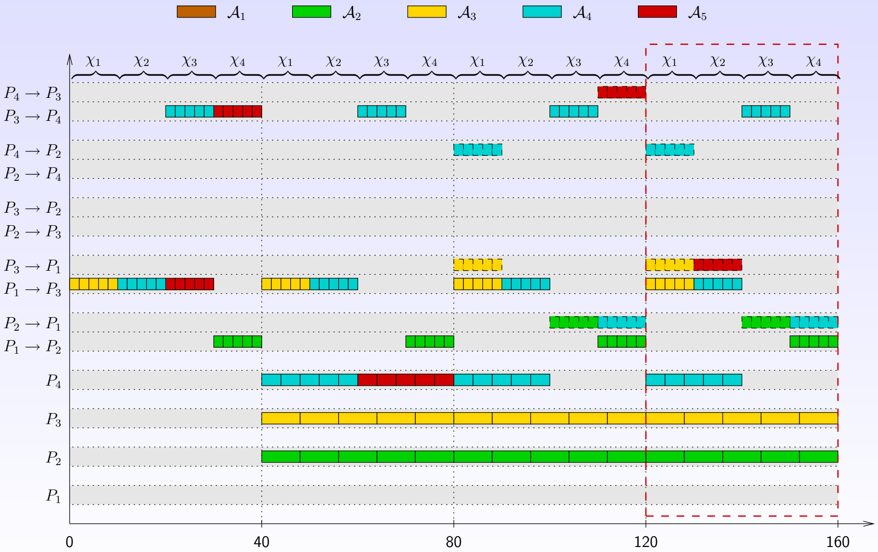
Cyclic scheduling achieving optimal throughput



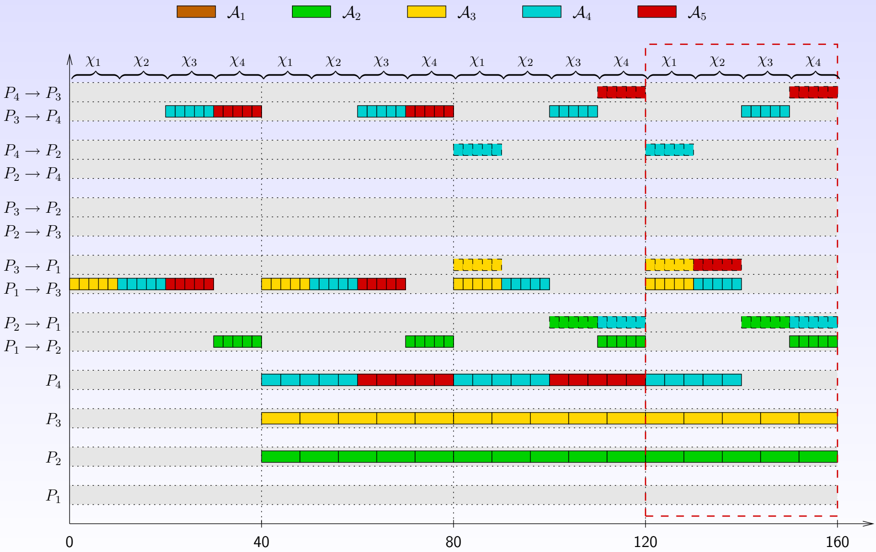
Cyclic scheduling achieving optimal throughput



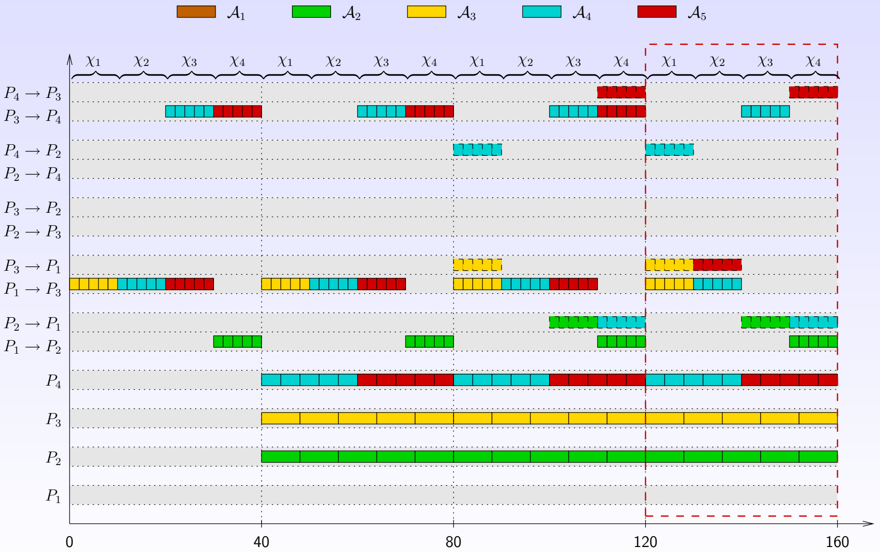
Cyclic scheduling achieving optimal throughput



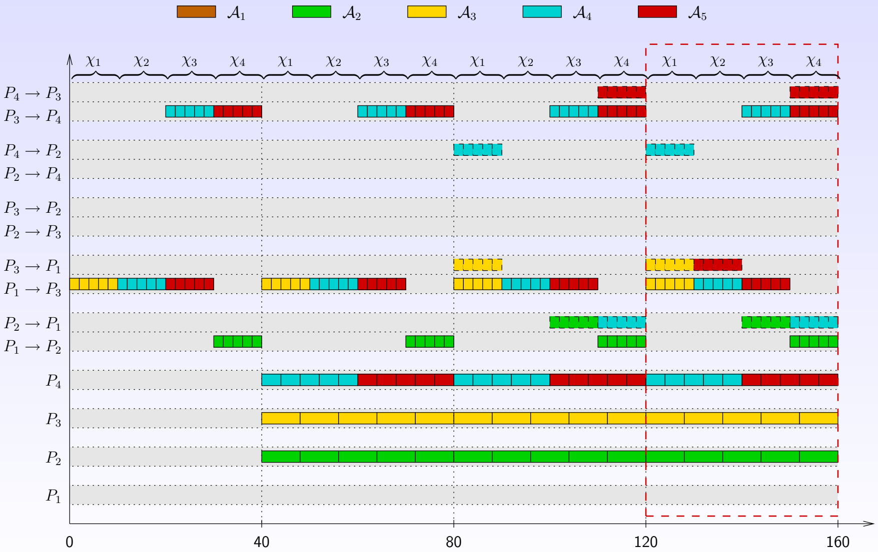
Cyclic scheduling achieving optimal throughput



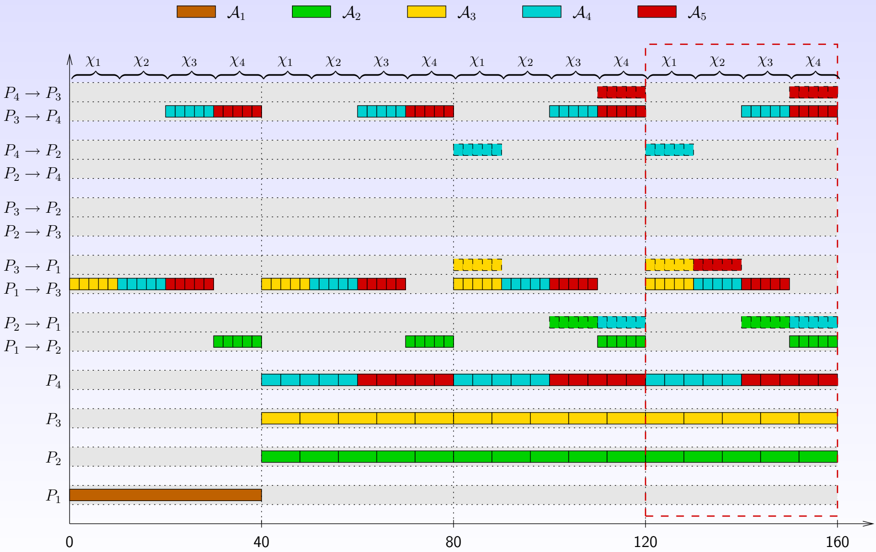
Cyclic scheduling achieving optimal throughput



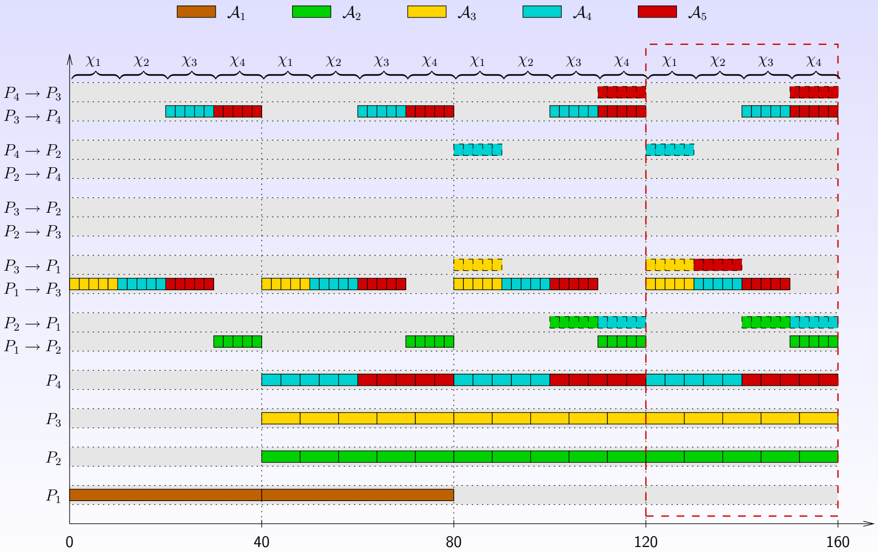
Cyclic scheduling achieving optimal throughput



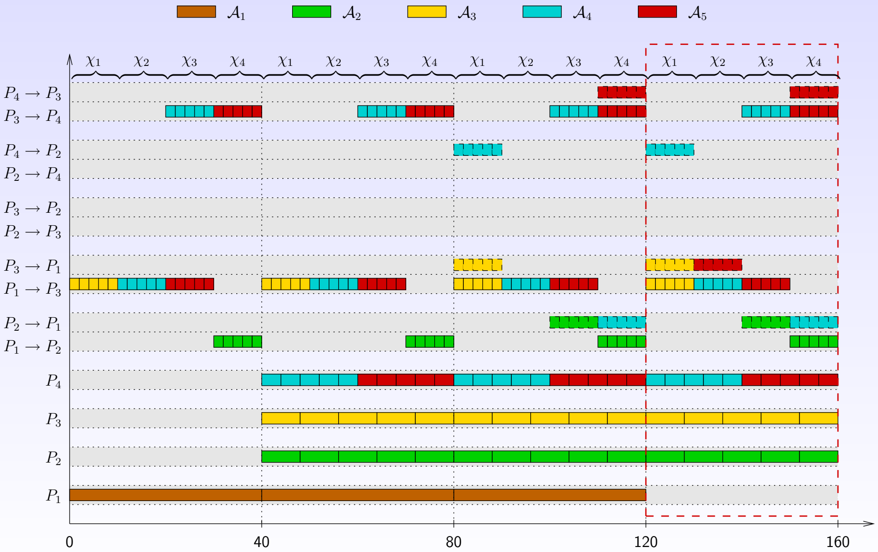
Cyclic scheduling achieving optimal throughput



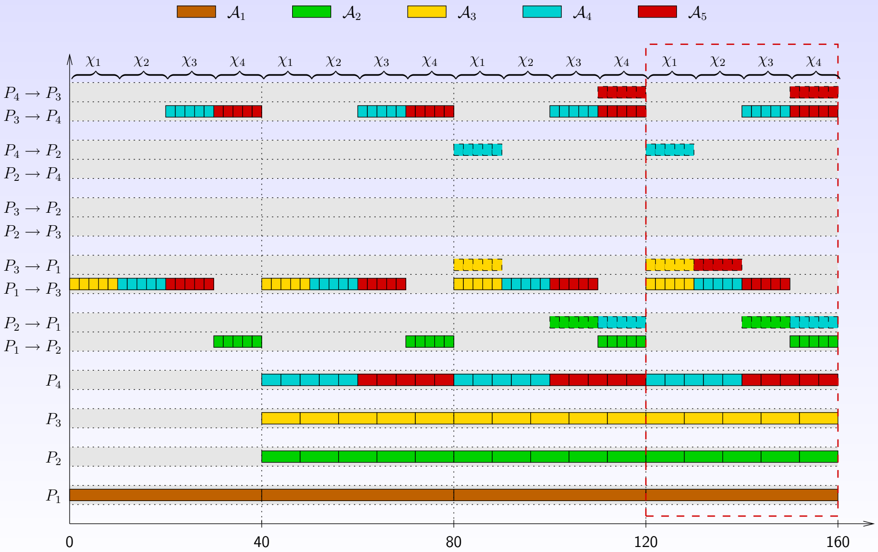
Cyclic scheduling achieving optimal throughput



Cyclic scheduling achieving optimal throughput



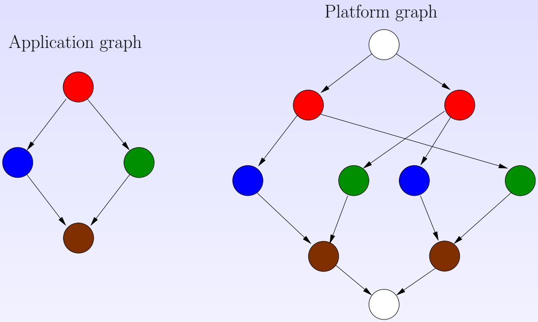
Cyclic scheduling achieving optimal throughput



Outline

Moving to general task graphs

What if there are dependencies?



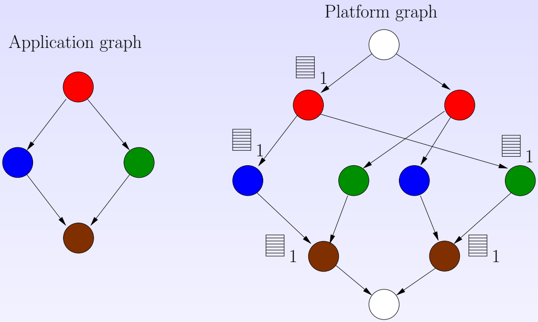
forget about dependencies
consider only activities } $\Rightarrow$ false solution!

In fact:

- NP-hard problem in the general case
- polynomial algorithm for bounded *dependency depth*

Moving to general task graphs

What if there are dependencies?



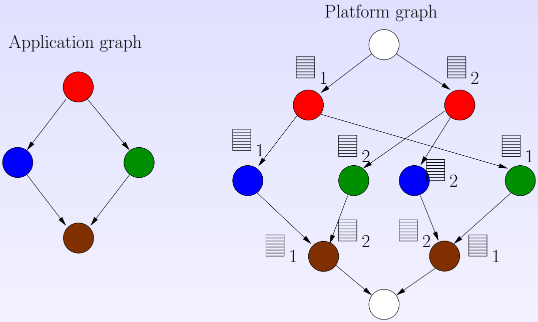
forget about dependencies
consider only activities
} ⇒ false solution!

In fact:

- NP-hard problem in the general case
- polynomial algorithm for bounded *dependency depth*

Moving to general task graphs

What if there are dependencies?



forget about dependencies
consider only activities

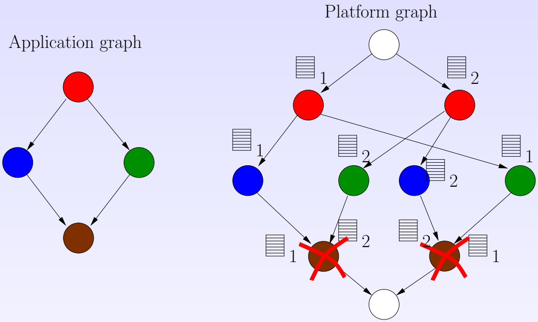
$\Rightarrow$ false solution!

In fact:

- NP-hard problem in the general case
- polynomial algorithm for bounded *dependency depth*

Moving to general task graphs

What if there are dependencies?



forget about dependencies
consider only activities
$\Big\}$ $\Rightarrow$ false solution!

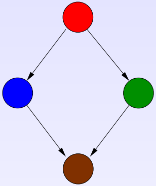
In fact:

- NP-hard problem in the general case
- polynomial algorithm for bounded *dependency depth*

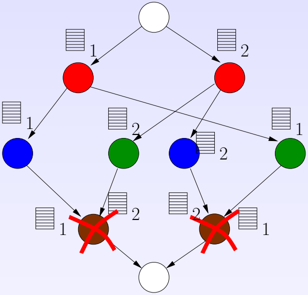
Moving to general task graphs

What if there are dependencies?

Application graph



Platform graph



$$\left.\begin{array}{c}\text{forget about dependencies}\\ \text{consider only activities}\end{array}\right\} \Rightarrow \text{false solution!}$$

In fact:

- NP-hard problem in the general case
- polynomial algorithm for bounded *dependency depth*

Outline

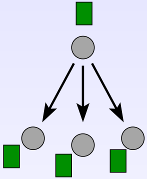
Collective communications

Collective communications: communications between more than 2 machines

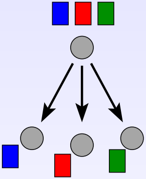
In the assumption of steady-state: pipelined communications

$\rightsquigarrow$ a large number of messages follow the same scheme

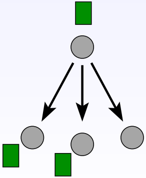
broadcast:



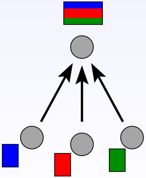
scatter:



multicast:



reduce:

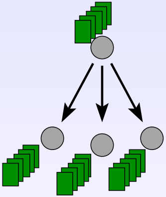


Collective communications

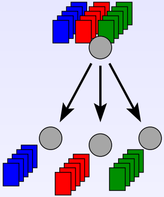
Collective communications: communications between more than 2 machines

In the assumption of steady-state: pipelined communications
$\rightsquigarrow$ a large number of messages follow the same scheme

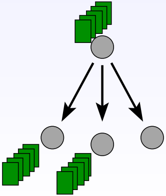
broadcast:



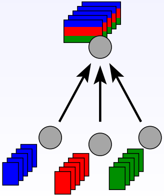
scatter:



multicast:



reduce:



Collective communications – Results

operation	allocations	complexity
broadcast	spanning tree	polynomial
scatter	set of paths	polynomial
gossip	set of paths	polynomial
reduce	*reduce tree*	polynomial
multicast	Steiner tree	NP-hard
parallel prefix	*parallel prefix tree*	NP-hard

For polynomial operations, we can derive an efficient approach to solve the problem under the bidirectional one-port model.

Collective communications – Results

operation	allocations	complexity
broadcast	spanning tree	polynomial
scatter	set of paths	polynomial
gossip	set of paths	polynomial
reduce	*reduce tree*	polynomial
multicast	Steiner tree	NP-hard
parallel prefix	*parallel prefix tree*	NP-hard

For polynomial operations, we can derive an efficient approach to solve the problem under the bidirectional one-port model.

Collective communications – Results

operation	allocations	complexity
broadcast	spanning tree	polynomial
scatter	set of paths	polynomial
gossip	set of paths	polynomial
reduce	*reduce tree*	polynomial
multicast	Steiner tree	NP-hard
parallel prefix	*parallel prefix tree*	NP-hard

For polynomial operations, we can derive an efficient approach to solve the problem under the bidirectional one-port model.

Collective communications – Results

operation	allocations	complexity
broadcast	spanning tree	polynomial
scatter	set of paths	polynomial
gossip	set of paths	polynomial
reduce	*reduce tree*	polynomial
multicast	Steiner tree	NP-hard
parallel prefix	*parallel prefix tree*	NP-hard

For polynomial operations, we can derive an efficient approach to solve the problem under the bidirectional one-port model.

Collective communications – Results

operation	allocations	complexity
broadcast	spanning tree	polynomial
scatter	set of paths	polynomial
gossip	set of paths	polynomial
reduce	*reduce tree*	polynomial
multicast	Steiner tree	NP-hard
parallel prefix	*parallel prefix tree*	NP-hard

For polynomial operations, we can derive an efficient approach to solve the problem under the bidirectional one-port model.

Collective communications – Results

operation	allocations	complexity
broadcast	spanning tree	polynomial
scatter	set of paths	polynomial
gossip	set of paths	polynomial
reduce	*reduce tree*	polynomial
multicast	Steiner tree	NP-hard
parallel prefix	*parallel prefix tree*	NP-hard

For polynomial operations, we can derive an efficient approach to solve the problem under the bidirectional one-port model.

Collective communications – Results

operation	allocations	complexity
broadcast	spanning tree	polynomial
scatter	set of paths	polynomial
gossip	set of paths	polynomial
reduce	*reduce tree*	polynomial
multicast	Steiner tree	NP-hard
parallel prefix	*parallel prefix tree*	NP-hard

For polynomial operations, we can derive an efficient approach to solve the problem under the bidirectional one-port model.

Outline

Steady state scheduling: good news and bad news

- ☺ Steady state scheduling: throughput maximization is much easier that makespan minimization and still realistic
- ☺ One-port model: first step towards designing realistic scheduling heuristics (other realistic models have been proposed in this context)
- ☺ Steady-state circumvents complexity of scheduling problems . . . while deriving efficient (often asymptotically optimal) scheduling algorithms

- ☹ Memory constraints, latency, period size may be large...
- ☹ Need to acquire a good knowledge of the platform graph (ENV, Alnem, NWS...)
- ☹ Taking into account changes in resource performances is still difficult: build super-steps and recompute optimal solution at the end of each super-step...

Steady state scheduling: good news and bad news

- ☺ Steady state scheduling: throughput maximization is much easier that makespan minimization and still realistic
- ☺ One-port model: first step towards designing realistic scheduling heuristics (other realistic models have been proposed in this context)
- ☺ Steady-state circumvents complexity of scheduling problems ...while deriving efficient (often asymptotically optimal) scheduling algorithms

- ☹ Memory constraints, latency, period size may be large...
- ☹ Need to acquire a good knowledge of the platform graph (ENV, Alnem, NWS...)
- ☹ Taking into account changes in resource performances is still difficult: build super-steps and recompute optimal solution at the end of each super-step...

Dynamic platforms

On large scale distributed systems:

- resource performances may change over time (resource sharing, node may appear and disappear)
- impossible to maintain a coherent snapshot of the platform at a given node and recompute optimal solution
- using fully greedy dynamic scheduling algorithms is known to lead to bad results
- inject some static knowledge into dynamic schedulers

Taking dynamic performances into account

Need for decentralized and robust scheduling algorithms based on static knowledge

What do robust and dynamic mean?

Need for metrics in order to analyze algorithms

Robust

- If $\rho_{\text{opt}}(t)$ denotes the optimal throughput for platform at time t and $T(N)$ denotes the time to process N tasks using proposed scheduling algorithm
- The objective is

$$\frac{N}{\displaystyle\int_{t=0}^{T(N)} \rho_{\text{opt}}(t)dt} \longrightarrow_{N \longrightarrow +\infty} 1$$

Decentralized

at any time step, a node makes its decisions according to

- its state (local memory)
- the states of its immediate neighbors

Outline

Fluid relaxation (cont'd!)

- Throughput maximization
  - concentrate on steady state
  - define activity variables
  - then, rebuild allocations and schedule

- Dynamic platforms:
  - put tasks in different queues
  - define potential functions associated to those queues
  - let tasks move "by themselves" from high to low potentials
  - areas where tasks are processed quickly will become low potential areas (tasks being removed)
  - areas where tasks are processed slowly will become high potential areas

Example: scheduling independent tasks

For the sake of simplicity, we will assume that

- that $\rho_{\min} = \min \rho_{\mathsf{opt}}(t)$ is known
- and we will prove that

$$\left(\frac{N}{T(N)}\right) \geq \rho_{\min}.$$

- We will also assume that the platform graph is a tree

(In fact, with more care and using a slightly different communication model, we could prove

$$N \geq \sum_{i=0}^{T(N)} \min_{t \in [i;i+1]} \rho_{\mathsf{opt}}(t)$$

for general platform graphs.)

Example: scheduling independent tasks

For the sake of simplicity, we will assume that

- that $\rho_{\min} = \min \rho_{\mathsf{opt}}(t)$ is known
- and we will prove that

$$\left(\frac{N}{T(N)}\right) \geq \rho_{\min}.$$

- We will also assume that the platform graph is a tree

  (In fact, with more care and using a slightly different communication model, we could prove

$$N \geq \sum_{i=0}^{T(N)} \min_{t \in [i;i+1]} \rho_{\mathsf{opt}}(t)$$

  for general platform graphs.)

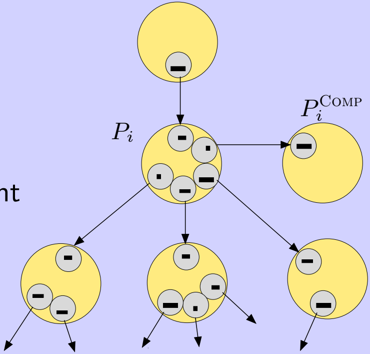
Queues (1)

Credits

based on an algorithm for multi-commodity flows (Awerbuch Leighton)

Queues at slave nodes

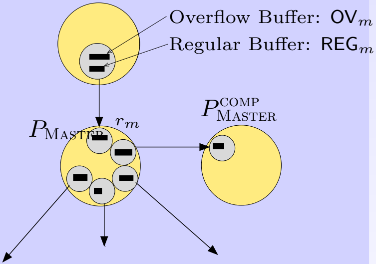
- each node P_i stores non processed tasks in N queues, where N denotes the children of P_i.
- each node P_i has a queue for incoming tasks (from its parent node)
- we introduce a fictitious processing neighbor node P_i^{COMP}.



Queues (2)

Queues at master node

- the master node is split into two parts (upper, lower).
- the upper master node holds a regular buffer and an overflow buffer
- the overflow buffer holds tasks that do not fit in the regular buffer
- the lower Master node works like any other node



Framework

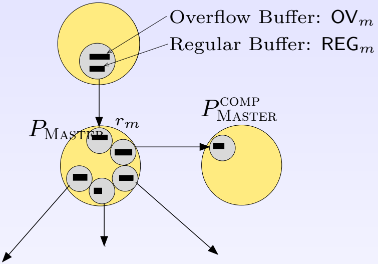
Queues and potential functions:

- Each queue (regular or overflow) is associated with an increasing (with the size of the queue) potential function
- The potential of an edge is the sum of queue potentials at the tail and head of the edge.
- The potential of a node is the sum of the potential of its outgoing edges.
- Nodes try to minimize their potential, given resource constraints (both processing power and 1-port).
- Thus, tasks go from high potential to low potential.

Potential functions

Potential functions at master node:

- The potential associated to the overflow buffer of size OV_m is $\sigma(\mathsf{OV}_m) = \mathsf{OV}_m \alpha e^{\alpha Q}$.
- The potential associated to the regular buffer of size REG_m is $\Phi(\mathsf{REG}_m) = e^{\alpha \mathsf{REG}_m}$.
- where α is a constant (depending on the network and the expected throughput) and Q is the maximal size of the regular buffer.

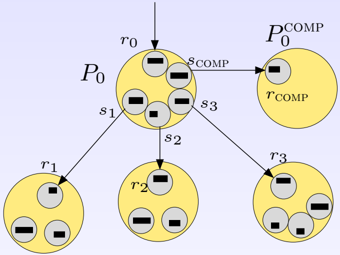


Potential functions

Potential functions at regular nodes:

- The potential associated to a regular buffer of size s is $\Phi(s) = e^{\alpha s}$.
- The potential associated to the edge (P_0, P_i) is $\Phi(P_0, P_1) = e^{\alpha s_i} + e^{\alpha r_i}$.
- The potential associated to the node P_0 is

$$\Phi(P_0) = \sum_i (e^{\alpha s_i} + e^{\alpha r_i}) + e^{\alpha s_{\text{COMP}}} + e^{\alpha r_{\text{COMP}}}$$



Overall Algorithm

Time is divided in rounds, each round consists in 4 steps.

Phase 1: At upper master node, add $(1-\varepsilon)\rho_{\min}$ units of tasks to the overflow queue. Then move as many tasks as possible from the overflow queue to the regular queue (given maximum height constraint)

Phase 2: At P_i, push flow across edges so as to minimize the potential of P_i without violating capacity constraints for each edge.

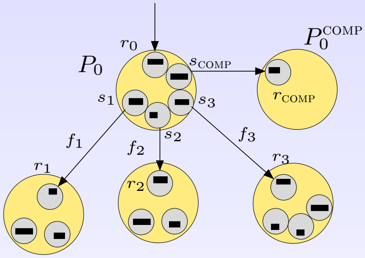
Phase 3: At P_i^{COMP}, empty the sink queue r_{COMP}

Phase 4: Re-balance each node P_i, so that the queues at P_i have same size.

Phase 2 detailed

How to minimize potential at P_i:

- The potential associated to the node P_0 is
  $$\Phi(P_0) = \sum_i \left(e^{\alpha s_i} + e^{\alpha r_i}\right) + e^{\alpha s_{\text{COMP}}} + e^{\alpha r_{\text{COMP}}}.$$



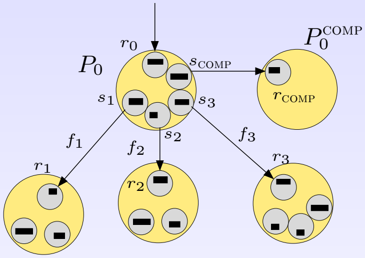
- Satisfying processing constraints is easy: do not send more than w_0 tasks.
  - Satisfying 1 port constraint:
    $$\text{Minimize } \sum_i \left(e^{\alpha(s_i - f_i)} + e^{\alpha(r_i + f_i)}\right)$$
    $$\begin{cases} f_i \geq 0 & \text{(directed edge)} \\ \sum f_i c_i \leq 1 & \text{(one port constraint)} \end{cases}$$

  Convex optimization problem, the f_i's can be determined using Karush-Kuhn-Tucker conditions.

Phase 2 detailed

How to minimize potential at P_i:

- The potential associated to the node P_0 is
  $$\Phi(P_0) = \sum_i \left(e^{\alpha s_i} + e^{\alpha r_i}\right) + e^{\alpha s_{\text{COMP}}} + e^{\alpha r_{\text{COMP}}}.$$
- Satisfying processing constraints is easy: do not send more than w_0 tasks.



- Satisfying 1 port constraint:

  Minimize $\sum_i \left(e^{\alpha(s_i - f_i)} + e^{\alpha(r_i + f_i)}\right)$

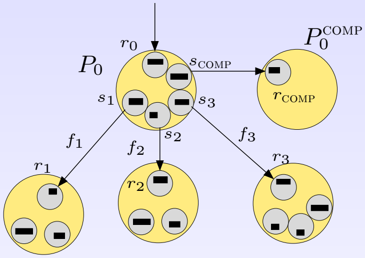
  $$\begin{cases} f_i \geq 0 & \text{(directed edge)} \\ \sum f_i c_i \leq 1 & \text{(one port constraint)} \end{cases}$$

  Convex optimization problem, the f_i's can be determined using Karush-Kuhn-Tucker conditions.

Phase 2 detailed

How to minimize potential at P_i:

- The potential associated to the node P_0 is
  $\Phi(P_0) = \sum_i \left(e^{\alpha s_i} + e^{\alpha r_i}\right)$
  $+ e^{\alpha s_{\text{COMP}}} + e^{\alpha r_{\text{COMP}}}$.
- Satisfying processing constraints is easy: do not send more than w_0 tasks.



- Satisfying 1 port constraint:

$$\text{Minimize } \sum_i \left(e^{\alpha(s_i - f_i)} + e^{\alpha(r_i + f_i)}\right)$$

$$\begin{cases} f_i \geq 0 & \text{(directed edge)} \\ \sum f_i c_i \leq 1 & \text{(one port constraint)} \end{cases}$$

Convex optimization problem, the f_i's can be determined using Karush-Kuhn-Tucker conditions.

Potential analysis during one round

- **Phase 1:** At upper master node, add $(1-\varepsilon)\rho_{\min}$ units of tasks to the overflow queue. Then move as many tasks as possible from the overflow queue to the regular queue (given maximum height constraint) $\nearrow$
- **Phase 2:** At P_i, push flow across edges so as to minimize the potential of P_i without violating capacity constraints for each edge. $\searrow$
- **Phase 3:** At P_i^{COMP}, empty the sink queue r_{COMP} $\searrow$
- **Phase 4:** Re-balance each node P_i, so that the queues at P_i have same size. $\searrow$

Potential $\nearrow$ during phase 1 can be evaluated easily
Potential $\searrow$ during phases 2-4 strongly depends on local queue sizes. . .

Potential analysis during one round

Sketch of the proof: Analyzing directly potential decrease during phase 2 is difficult, but

- we know that there exists a solution with throughput $\rho_{\min}$
- since potential minimization is optimal (given resource constraint) during Phase 2
- $\Longrightarrow$ the potential decrease during Phase 2 is at least the potential decrease that would be induced by the solution with throughput $\rho_{\min}$
- the potential decrease that would be induced by the solution with throughput $\rho_{\min}$ can be determined easily

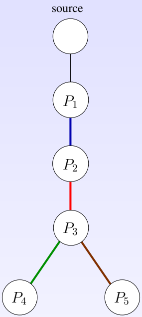
$\Longrightarrow$ we get a lower bound for potential decrease during Phase 2 (and neglect potential decreases during Phases 3-4)

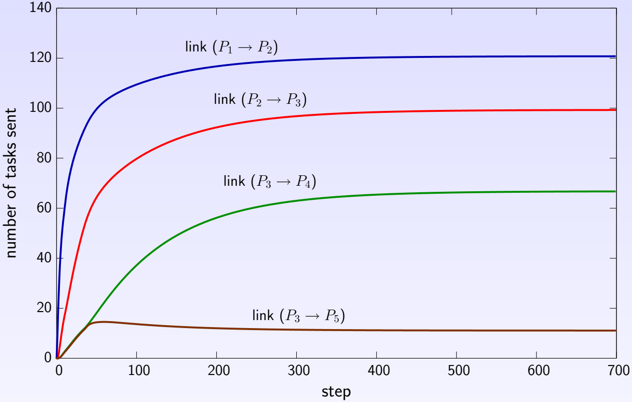
Sketch of the proof (end!)

- Using above technique, we can prove that the overall potential remains bounded
- $\Longrightarrow$ the overall number of non-processed tasks in the network remains bounded
- Since we inject $(1 - \varepsilon)\rho_{\min}$ tasks at each round, this means that almost all tasks have been processed

$\Longrightarrow$ the overall throughput is optimal
(almost, due to ε, that can be chosen arbitrarily small)

Small example – convergence





- independent tasks
- no result files sent back to source

Conclusion on dynamic solutions

- Deriving efficient dynamic solutions for unstable environments is still a wide area of research
- For the very simple problem we looked at (independent tasks on a tree):
  - how to determine the minimal throughput (one solution may be to look at queue sizes)?
  - how to move from fractional task numbers to actual tasks (consider larger rounds and round results)?
  - what happens if performances change during one round, and things get de-synchronized?
- But we feel that, using such solutions, it is possible to achieve much better results than with purely dynamic schedulers

☺ Still plenty of work!