Scheduling on clusters and grids

Grégory Mounié, Yves Robert et **Denis Trystram**

ID-IMAG

6 mars 2006

Basic references

- Chapitre 3 (Gestion de ressources) "Informatique Répartie", Trystram, Slimani et Jemni editeurs, Hermes, 2005.
- Joseph Leung "Handbook of Scheduling", Chapman & Hall, 2004

Traditional scheduling – Framework

Application = DAG $G = (T, E, p)$

- $\mathcal{T}$ = set of tasks
- E = precedence constraints
- $p(T)$ = computational cost of task T (execution time)
- $c(T, T')$ = communication cost (data sent from T to T')

Platform

- Set of p (identical) processors

Schedule

- $\sigma(T)$ = date to start the execution of task T
- $\pi(T)$ = processor assigned to it

Traditional scheduling – Constraints

Data dependencies If $(T, T') \in E$ then

- if $\pi(T) = \pi(T')$ then

$$\sigma(T) + p(T) \leq \sigma(T')$$

- if $\pi(T) \neq \pi(T')$ then

$$\sigma(T) + p(T) + c(T, T') \leq \sigma(T')$$

Resource constraints (sequential tasks)

$$\pi(T) = \pi(T') \Rightarrow [\sigma(T), \sigma(T) + p(T)[\bigcap [\sigma(T'), \sigma(T') + p$$
$$= \emptyset$$

Traditional scheduling – Objective functions

Makespan or total execution time

$$C_{max}(\sigma) = \max_{T \in \mathcal{T}} (\sigma(T) + p(T))$$

Other classical objectives :

- Sum of completion times (with its weighted variant)
- With arrival times : maximum flow (response time), or sum flow
- More oriented to fair solutions : maximum stretch, or sum stretch

Scheduling problem

Computational units are identifed and their relations are analyzed.

Scheduling

Determine when and where computational units will be executed.

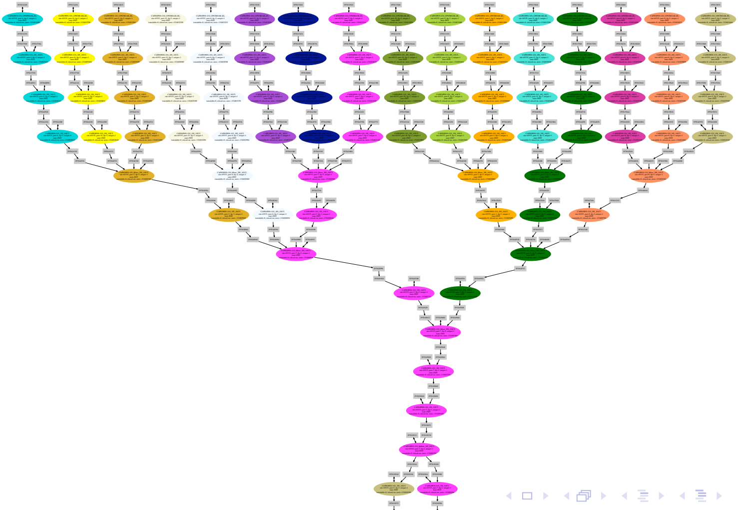
Precedence Task Graph

Let $G = (V, E)$ be a weighted directed acyclic graph iff (partial order)

- The vertices are weighted by the execution times.
- The arcs are weighted by the data to be transfered from a task to another.

Notice that some colleagues are considering variants (bipartite, Data flow, etc..)

Precedence Task Graph



Formal Definition

The problem of scheduling graph $G = (V, E)$ weighted by function p on m processors : (without communication) Determine the pair of functions (σ, π) subject to the respect of precedences :
$\forall (i,j) \in E : \sigma(j) \geq \sigma(i) + p(i, \pi(i))$
Usual objective : to minimize the makespan (C_{max})

Theorem

Minimizing the makespan is NP-Hard [Ullman 75]

One step further

This problem remains NP-Hard even in relaxed cases : (independent tasks, trees, etc.)

Consequence

We have to find "efficient" heuristics

Evaluation

The evaluation of a heuritic for a criterion ω.

Definition (Competitive Ratio)

A real number ρ such that $\forall$ instance $\mathcal{I}$, $\omega(\mathcal{I}) = r(\mathcal{I})\omega^*(\mathcal{I})$ with $\rho = sup(r(\mathcal{I}))$

Lower bounds

Basic tool : Theorem of impossibility [Lenstra-Shmoys'95]

Given a scheduling problem and an integer c, if it is NP-complete to schedule this problem in less than c times, then there is no schedule with a competitive ratio lower than (c+1)/c.

Application

Theorem

The problem of deciding (for any UET graph) if there exists a valid schedule of length at most 3 is NP-complete.

Démonstration.

by reduction from CLIQUE □

Consequence

It is impossible to find a heuristic better than 4/3 iff $P \neq NP$

Lower Bounds

We are looking for simple algorithms that have good competitive ratios.
List scheduling is such a nice framework.

List scheduling

- *Initialization :*
  1. Priority queue = list of free tasks (tasks without predecessors) sorted by priority
  2. t is the current time step : $t = 0$.
- *While it remains some tasks to execute* :
  1. Add new free tasks, if any, to the queue. If the execution of a task terminates at time step t, suppress this task from the predecessor list of all its successors. Add those tasks whose predecessor list has become empty.
  2. If there are q available processors and r tasks in the queue, remove first $\min(q, r)$ tasks from the queue and execute them ; if T is one of these tasks, let $\sigma(T) = t$.
  3. Increment t.

List scheduling

- Priority level (off-line)
  - Use critical path : longest path from a task with no predecessor to an exit node
  - Computed recursively by a bottom-up traversal of the graph
- Implementation details
  - Cannot iterate from $t = 0$ to $t = C_{max}(\sigma)$ (exponential in problem size)
  - Use a heap for free tasks valued by priority level
  - Use a heap for processors valued by termination time
  - Complexity $O(|V| \log |V| + |E|)$

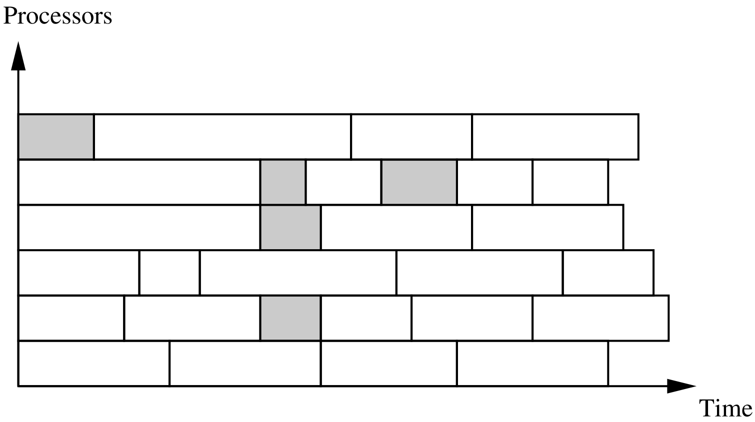
Analysis of List scheduling

Theorem

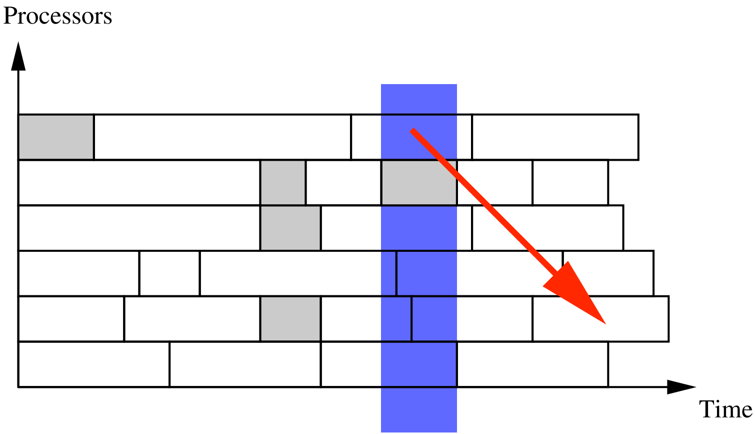
Performance guarantee of list scheduling

$$C_{max}(\sigma) \leq C^*_{max}(2 - \frac{1}{m})$$

Analysis

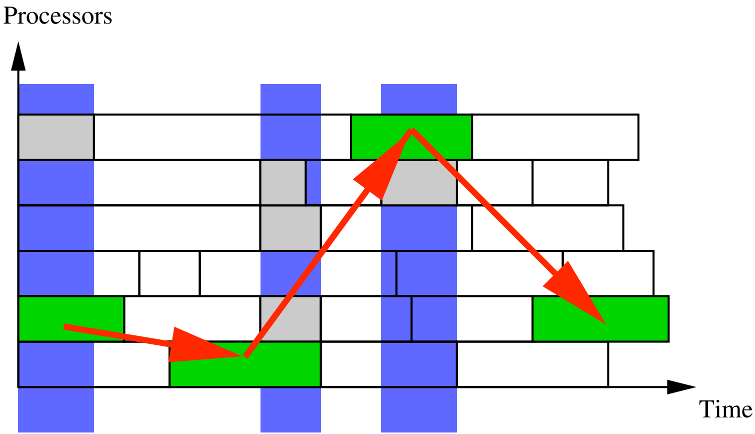


Analysis



Interval with idle time

Analysis



Intervals with idle time

Analysis

Theorem (List scheduling analysis)

$$C_{max}(\sigma) = \frac{W + Idle}{m}$$

where

- $\frac{W}{m} \leq C^*_{max}$
- at most $(m-1)$ idle processors
  - $Idle \leq (m-1)T_\infty$
  - $T_\infty \leq C^*_{max}$

Corollary

$$C_{max}(\sigma) \leq (2 - \frac{1}{m})C^*_{max}$$

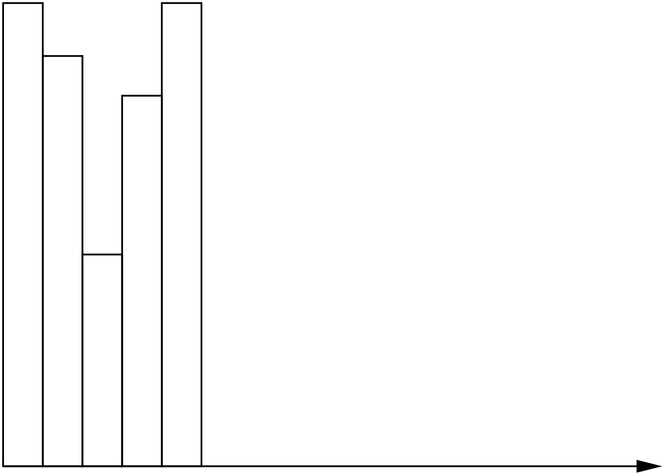
Brent's Lemma

Lemma (Brent)

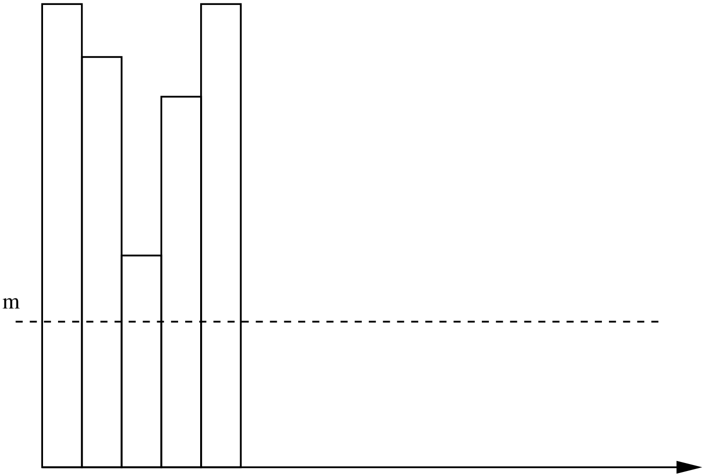
Let ρ be the competitive ratio of an algorithm with an unbounded number of processors. There exists an algorithm with performance ratio 2ρ for an arbitrary number of processors.

Since there exists an optimal algorithm for scheduling a graph with unbounded number of processors, there is a 2-approximation algorithm for m fixed (this is another way for looking at the graham's bound).

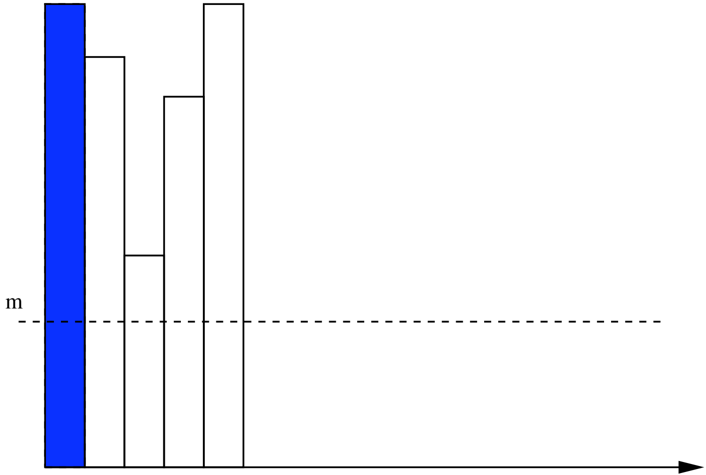
Brent's Lemma



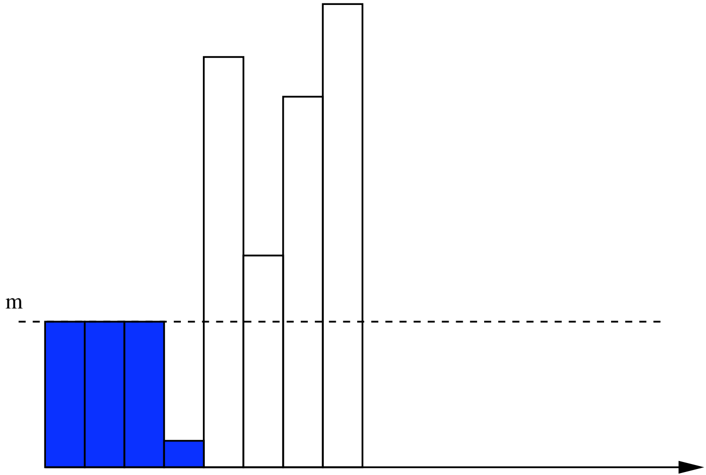
Brent's Lemma



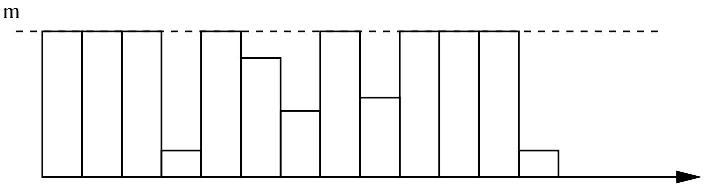
Brent's Lemma



Brent's Lemma



Brent's Lemma



Brent's Lemma

The proof is quite similar to Graham's analysis

Démonstration.

$$C_{max} = \sum^{C_{max}^{\infty}} \lceil \frac{Work(t)}{m} \rceil$$

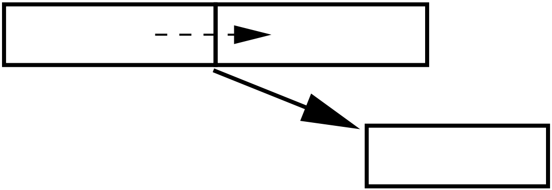
thus (in a Graham's way)

$$C_{max} \leq C_{max}^{\infty} + \sum^{C_{max}^{\infty}} \lfloor \frac{Work(t)}{m} \rfloor$$

□

Delay model

Let us consider a task with two successors.



Complexity : This model is more complicated than the central scheduling problem (Lenstra et al. 1990).
Scheduling a graph with communication on a unbounded number of processors is NP-hard.

List scheduling – With communications

ETF *Earliest Task First*

- Dynamically recompute priorities of free tasks
- Select free task that finishes execution first (on best processor), given already taken scheduling decisions
- Higher complexity $O(|V|^3 p)$
- May miss “urgent” tasks on the critical path

There exists a performance guaranty.
No efficient algorithm is known for large communication delays.

Other approaches

Two-steps : clustering + load balancing :

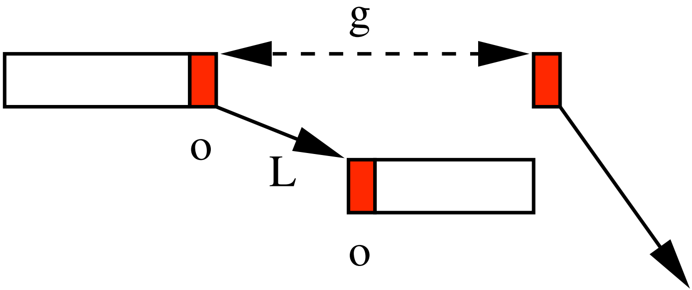
- DSC Dominant Sequence Clustering $O((|V| + |E|) \log |V|)$
- LLB List-based Load Balancing $O(C \log C + |V|)$ (C number of clusters generated by DSC)

HEFT : Heterogeneous Earliest Finishing Time

1. Priority level :
   - $\text{rank}(T_i) = \overline{w_i} + \max_{T_j \in \text{Succ}(T_i)}(\overline{\text{com}_{ij}} + \text{rank}(T_j))$, where $\text{Succ}(T)$ is the set of successors of T
   - Recursive computation by bottom-up traversal of the graph
2. Allocation
   - For current task T_i, determine best processor P_q : minimize $\sigma(T_i) + w_{iq}$
   - Enforce constraints related to communication costs
   - Insertion scheduling : look for $t = \sigma(T_i)$ s.t. P_q is available during interval $[t, t + w_{iq}[$
3. Complexity : same as MCP without/with insertion

LogP

The goal is to take into account local communication overhead.
Such computations are costly on network with high throughput.



Fork tree scheduling in the delay model

Fork tree

Instance : A fork tree task graph. The communication cost are modelized with the delay model. Unbounded number of processors.

Problem : minimizing C_{max}

Theorem (Fork Tree)

The scheduling of a fork tree is solvable in polynomial time.

Fork tree scheduling
in the LogP model

Fork tree - LogP

Instance : A fork tree task graph. The communication cost are modelized with the LogP model. Unbounded number of processors.

Problem : minimizing C_{max}

Theorem

The scheduling of a fork tree is NP-Hard.

About LogP

Fact (LogP)

A modelisation closer to reality induces often an increased complexity and a worse approximability.

Consequences

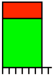
Alternative approaches are required to be able to schedule accurately and efficiently parallel applications.

Parallel Tasks Model

Set of independent jobs (Parallel Tasks).

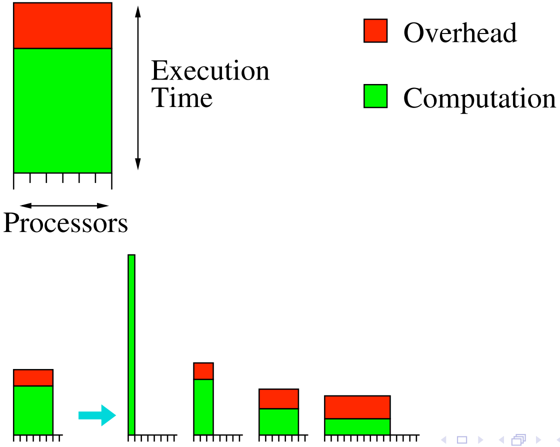
Parallel Tasks Model

Set of independent jobs (Parallel Tasks).



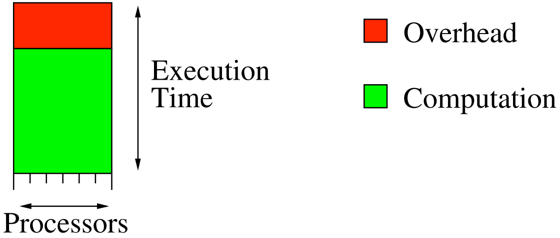
Parallel Tasks Model

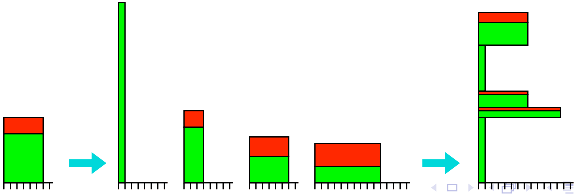
Set of independent jobs (Parallel Tasks).



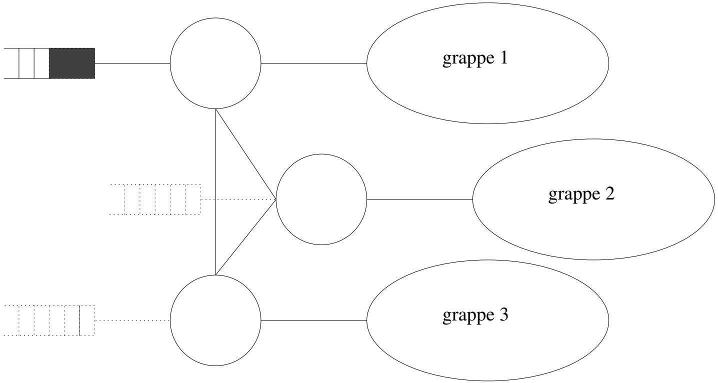
Parallel Tasks Model

Set of independent jobs (Parallel Tasks).

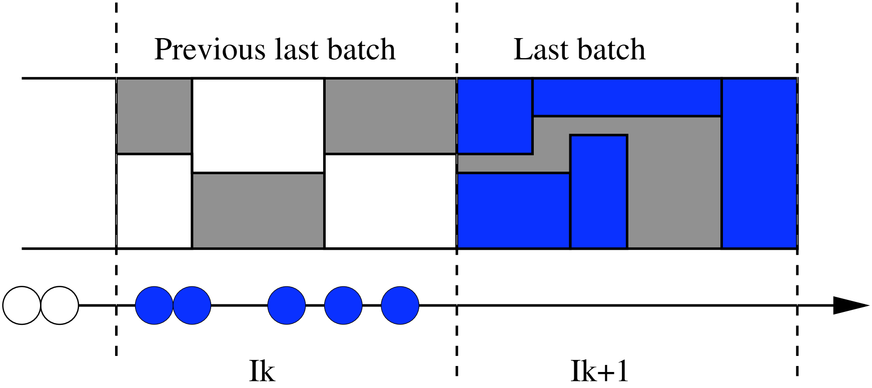




Batch scheduler principle



Analysis of batch scheduling



All tasks arrived in time interval I_k are scheduled in time interval I_{k+1}

Analysis of batch scheduling

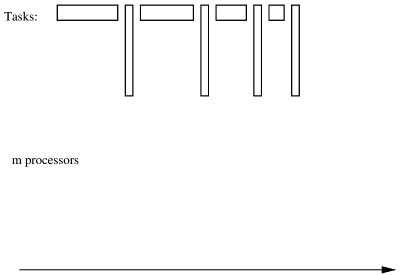
Theorem (Online batch scheduling)

On-line (batch) scheduling [Shmoys et al. – SIAM'95] : the approximation ratio is multiplied by a factor of 2.

All tasks arrived in time interval I_k can not be schedule before the beginning of I_k. Interval I_k and I_{k+1} are both smaller than ρC^*_{max}.

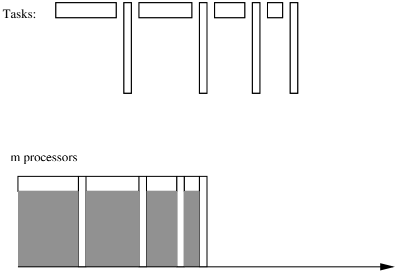
FIFO and co. principle

- Fifo guaranties no starvation. But, it may induced large idle time (m times worse than the optimal)



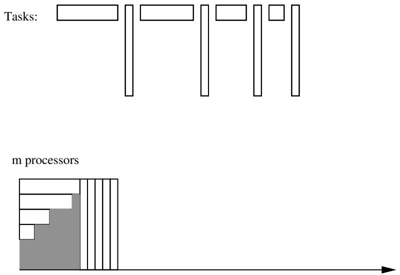
FIFO and co. principle

- Fifo guaranties no starvation. But, it may induced large idle time (m times worse than the optimal)



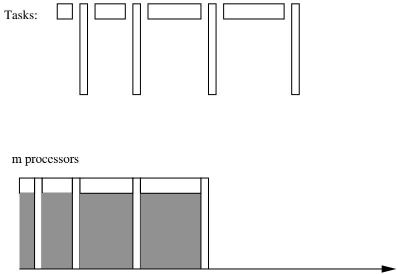
FIFO and co. principle

- Fifo guaranties no starvation. But, it may induced large idle time (m times worse than the optimal)
- Fifo with basic **back filling** is better



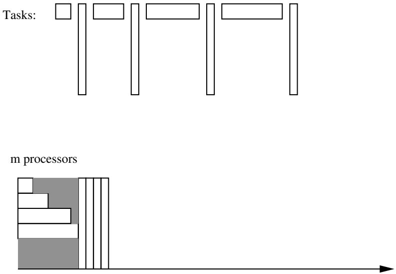
FIFO and co. principle

- Fifo guaranties no starvation. But, it may induced large idle time (m times worse than the optimal)
- Fifo with basic **back filling** is better but the worst case is the same (m times worse than the optimal)



FIFO and co. principle

- Fifo guaranties no starvation. But, it may induced large idle time (m times worse than the optimal)
- Fifo with basic **back filling** is better but the worst case is the same (m times worse than the optimal)
- Fifo with **aggressive** back filling is a list scheduling algorithm (less than twice the optimal)



Divisible load applications

DEFINITION : Divisible load applications can be divided into any number of independent pieces. Perfectly parallel job : any sub-task can itself be processed in parallel, and on any number of workers. This model is a good approximation for applications that consist of very (very) large numbers of identical, low-granularity computations

Continuous solution

Fact

Continuous solution for master-worker problem Instead of looking for integer solution, fractional solutions are found easily to solve optimally distribution of tasks on heterogeneous processors to minimize C_{max}
Heterogeneous processors, variants of topological networks, etc..

Heavy - but straightforward - techniques.

Continuous solution

Solving the problem with complex network or multiple send-receive per node is much more difficult

steady-state scheduling

- Communication in grid may be fluidised in continuous streams between nodes, in a steady state, neglecting initialisation and clean-up.
- The full detailed order does not need to be computed

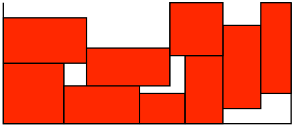
Point of view

Changing the point of view (eg. relaxing makespan) may help to get polynomial results in large scale applications

Mixed models

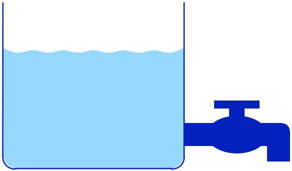
Best effort jobs

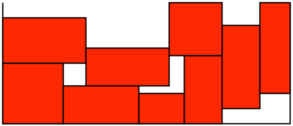
Easy to handle practically by filling the idle slots of a schedule.



Mixed models
Best effort jobs

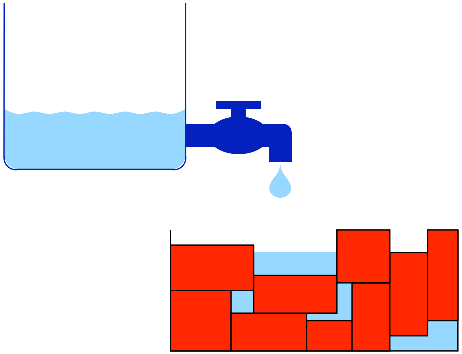
Easy to handle practically by filling the idle slots of a schedule.





Mixed models
Best effort jobs

Easy to handle practically by filling the idle slots of a schedule.



Do not feel alone...

Working group MAO of the GdR ASR (next meeting the 23rd march, LIP6, Paris).

Hot topics

- Dealing with incertainties and disturbances.

Hot topics

- Dealing with incertainties and disturbances.
- Game Theory and economical approaches for an alternative way of managing complex decisions.