Fairness issues in new large scale parallel platforms.

Denis TRYSTRAM

LIG – Université de Grenoble Alpes – Inria
Institut Universitaire de France

july 15, 2015

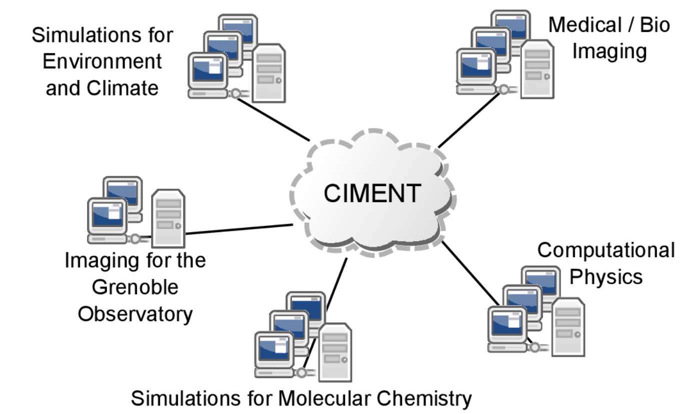
New challenges from e-Science

The scientific community has today the unprecedented ability to combine various computational resources into a powerful distributed system capable of analyzing massive data sets.

The main challenge is to allocate efficiently such jobs to the available resources.

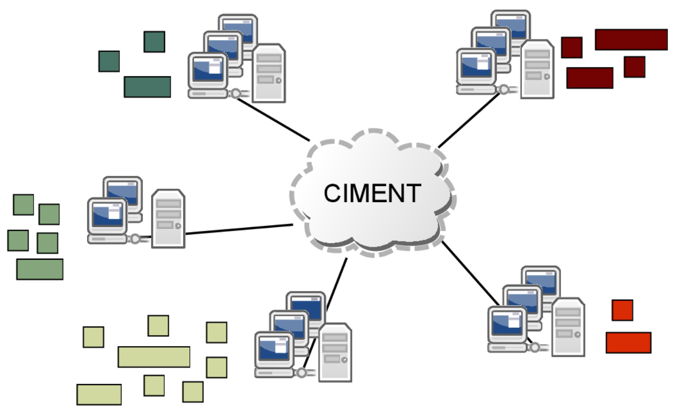
Example: An e-Science platform in Grenoble

Several labs issued from various communities share their computing resources...



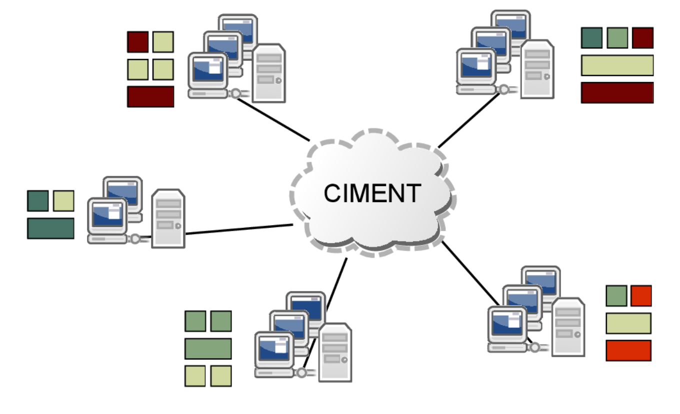
Example: An e-Science platform in Grenoble

Several labs issued from various communities share their computing resources...



Example: An e-Science platform in Grenoble

Several labs issued from various communities share their computing resources...



CiGRI: Each site has its own particular objective

Molecular Chemistry

Chemists are interested in obtaining the results of their simulations as fast as possible.
Objective: to minimize the maximum completion time

Medical analysis by bio-Imaging

Doctors are interested in delivering results of medical imaging analysis.
Objective: to minimize the average completion time or throughput

Ph.D students

Tuning an academic program for a delivery in a given deadline.
Objective: to minimize the completion time of a part (say 10%) of their jobs

CiGRI: Each site has its own particular objective

Molecular Chemistry

Chemists are interested in obtaining the results of their simulations as fast as possible.
Objective: to minimize the maximum completion time

Medical analysis by bio-Imaging

Doctors are interested in delivering results of medical imaging analysis.
Objective: to minimize the average completion time or throughput

Ph.D students

Tuning an academic program for a delivery in a given deadline.
Objective: to minimize the completion time of a part (say 10%) of their jobs

CiGRI: Each site has its own particular objective

Molecular Chemistry

Chemists are interested in obtaining the results of their simulations as fast as possible.
Objective: to minimize the maximum completion time

Medical analysis by bio-Imaging

Doctors are interested in delivering results of medical imaging analysis.
Objective: to minimize the average completion time or throughput

Ph.D students

Tuning an academic program for a delivery in a given deadline.
Objective: to minimize the completion time of a part (say 10%) of their jobs

CiGRI: Each site has its own particular objective

Molecular Chemistry

Chemists are interested in obtaining the results of their simulations as fast as possible.
Objective: to minimize the maximum completion time

Medical analysis by bio-Imaging

Doctors are interested in delivering results of medical imaging analysis.
Objective: to minimize the average completion time or throughput

Ph.D students

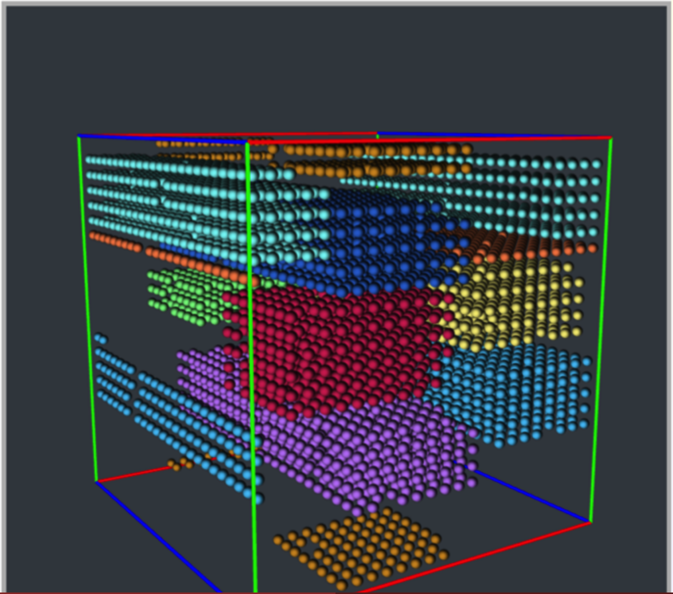
Tuning an academic program for a delivery in a given deadline.
Objective: to minimize the completion time of a part (say 10%) of their jobs

Another context: large scale HPC platforms

Sometimes various communities (users) share the same computing parallel platform.

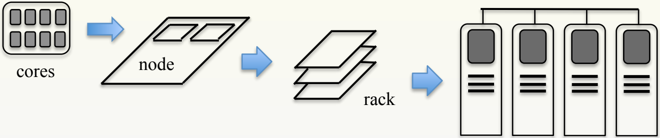
Multi-user scheduling

Jobs are submitted by campaigns by multiple users who are competing against each others for the available computing resources.



Motivation

Most available HPC platforms are hierarchical clusters.



- To present several important problems involving cooperation.
- To look at some algorithmic issues.

Objective of this talk

To investigate several facets of the **rules that govern how different participants engage in cooperation**.

We will show how to use scheduling algorithms to ensure efficient use of resources when cooperation takes place in several situations including:

- Classical systems without any local cooperation (pure centralized control)
- Forced cooperation between organizations that cannot be completely trusted
- Fairness among users

Main milestones

Key parameters:

- **Jobs:** sequential workflows, parallel (rigid, moldable, malleable), divisible loads
- **Resources:** identical, uniform hierarchical, heterogeneous
- **Objective:** minimize max of C_i (called makespan), mean flow time (ΣC_i), weighted versions, flow, stretch, ...
- off-line or on-line

C_i denotes the completion time of job i.

The simplest case

- **Jobs: sequential workflows**, parallel (rigid, moldable, malleable), divisible loads
- **Resources: identical** , uniform hierarchical, heterogeneous
- **Objective:** minimize **max of C_i (makespan)**, mean flow time (ΣC_i).

Schedule n independent jobs on m identical processors, aiming at minimizing the maximum completion time $Cmax$.

A magical recipe: list scheduling

Principle:
List algorithms are based on a list of ready jobs [Graham in 69]. As soon as there are available resources (processors), we allocate ready jobs.
This algorithm has a constant approximation guarantee of 2 in the worst case.

Remarks:

- List is a low cost algorithm (linear in the number of jobs).
- It is asymptotically optimal for a large number of jobs
- It works for both off-line and on-line settings.

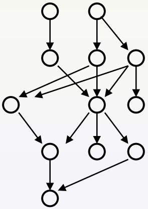
What about parallel jobs?

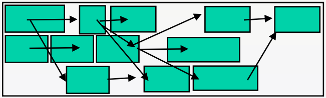
- **Jobs:** sequential workflow, **parallel rigid** or malleable, divisible loads
- **Resources:** **identical** , uniform hierarchical, heterogeneous
- **Objective:** Again, minimize the **makespan**, mean flow time (ΣC_i).

(multiple) Strip packing problems.

Rigid jobs

Rigid jobs correspond to parallel applications (where the number of processors is fixed like MPI programs).





Algorithms for one strip

Existing results (upper bounds)

- FCFS: arbitrarily bad
- List Scheduling is still a $(2 - \frac{1}{m})$-approximation for non-continuous case only! Introduced by Graham-Garey in 1975.
- Steinberg or Schiermeyer: fast 2-approximation.
- Jansen: very costly $(\frac{3}{2} + \epsilon)$-approximation.

Extension for multiple strips

The problem is completely solved now.

More sophisticated analysis, but the main point is that the bound is 2 instead of $\frac{3}{2}$.

Flavor of a centralized efficient algorithm.

Use a decomposition of the input (High jobs L_H, long and extra long jobs (L and XL) and the rest) and design algorithm which respects the structure of an optimal schedule:

Flavor of a centralized efficient algorithm.

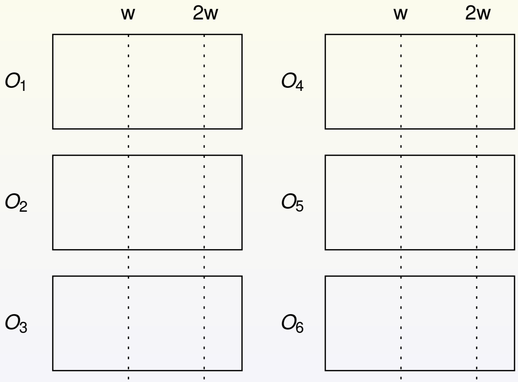
Use a decomposition of the input (High jobs L_H, long and extra long jobs (L and XL) and the rest) and design algorithm which respects the structure of an optimal schedule:

Topological properties

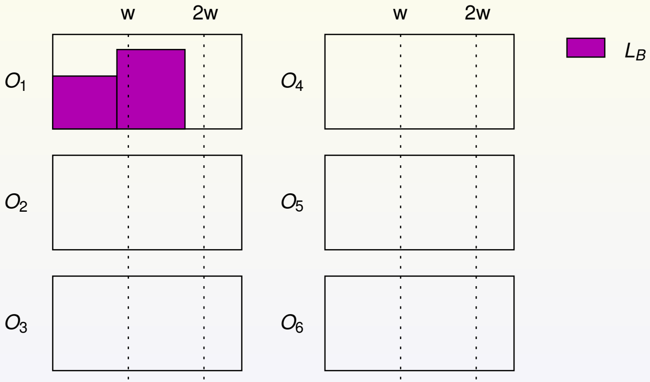
- $P(L_H) \leqslant N\omega$
  Only one “high” at any time instant on a cluster
- $Q(L_{XL} \bigcup L_L) \leqslant Nm$
  Only one “long” on any processor
- $S(I') \leqslant Nm\omega$
  All the jobs fit in the optimal

We target a $\frac{5}{2}$-approximation using a dual approximation scheme.

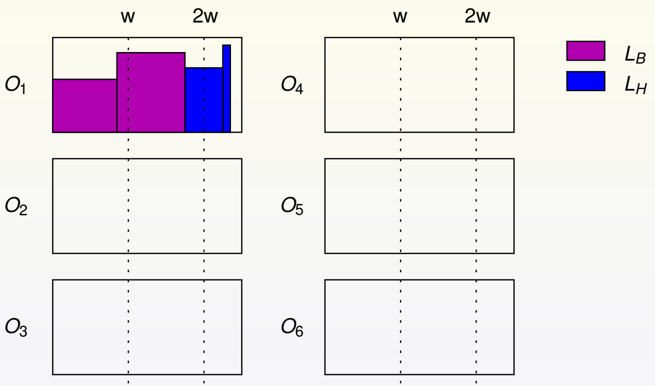
Running the algorithm (first steps...)



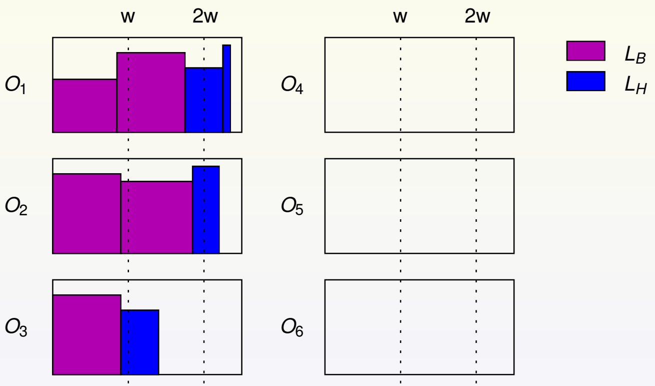
Running the algorithm (first steps...)



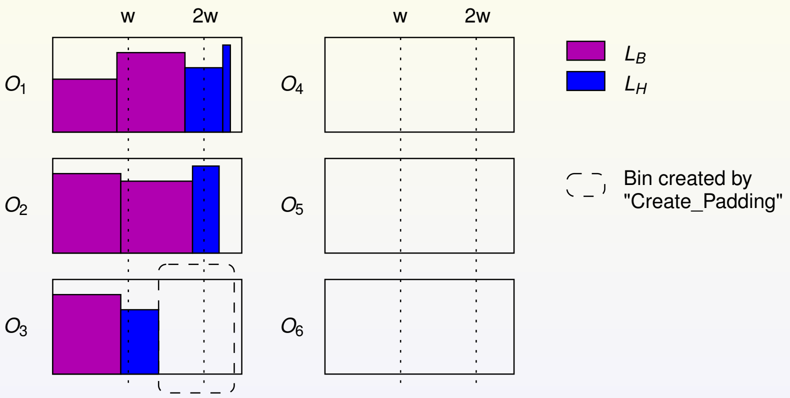
Running the algorithm (first steps...)



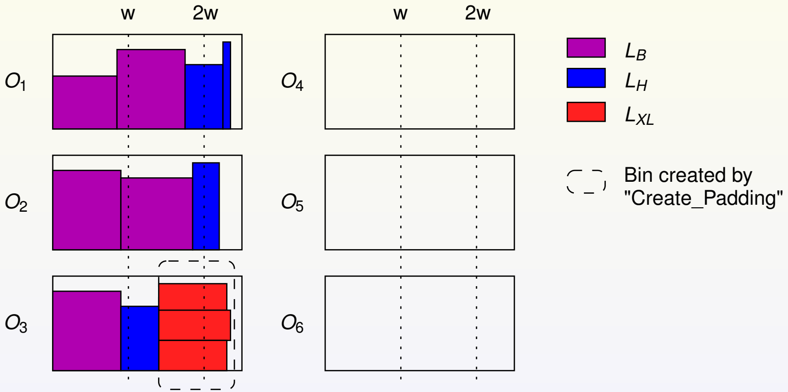
Running the algorithm (first steps...)



Running the algorithm (first steps...)



Running the algorithm (first steps...)

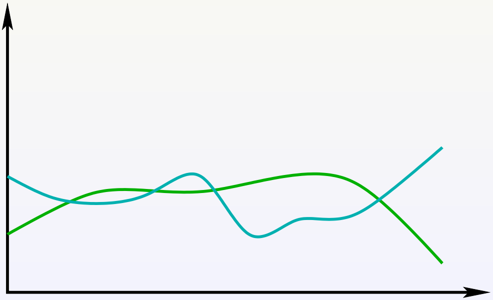


Multiple organizations: multiple strip packing

Motivation: Share computing power to dampen peaks (centralized control).

N clusters of *m* identical processors each. This number may also be different.

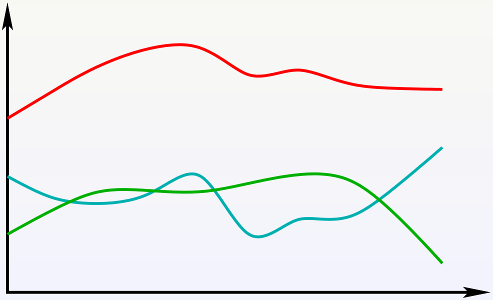
The inapproximation bound is 2 (proof by a Gap reduction).



Multiple organizations: multiple strip packing

Motivation: Share computing power to dampen peaks (centralized control).
N clusters of m identical processors each. This number may also be different.

The inapproximation bound is 2 (proof by a Gap reduction).



Outline

1 Multi-organization

2 Fairness issues and solution

3 Concluding remarks

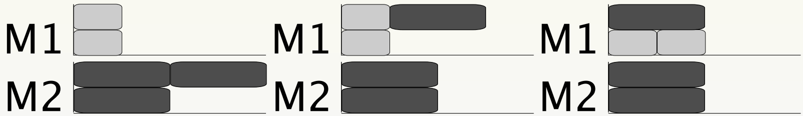
Model of multi-organization scheduling

- *organizations* $O^{(u)}$ have *resources* (clusters) and some *local jobs* $\{J_i^{(u)}\}$
- system goal: global makespan $C_{\max}$
- each organization minimizes the makespan of its local jobs $C_{\max}^{(u)} = \max_i C_i^{(u)}$
- idea: move jobs across clusters to optimize $C_{\max}$

Multi-objective optimization based on constraints on organizations' objectives

- an organization can not increase its local makespan $C_{\max}^{(u)}$ by cooperating with others
- schedule jobs locally (with makespan $C_{\max}^{(u)}(loc)$)
- optimization: $\min \max C_{\max}^{(u)}$ subject to $\forall u : C_{\max}^{(u)} \leqslant C_i^{(u)}(loc)$

Local constraints lead to a 3/2 lower bound on the global makespan



(a) local scheduling (b) globally-optimal with constraints (c) globally-optimal, no constraints

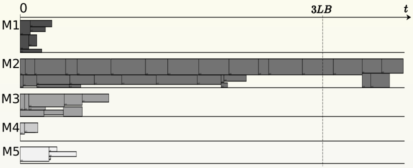
MOSP is NP-hard in the strong sense.

Outline of the scheduling algorithm (MOCCA)

3-approximation of the global makespan; local constraints are not violated
[P.-F. Dutot, F. Pascual, K. Rzadca, D. Trystram, IEEE TPDS 2011]

1. schedule jobs locally using highest-first (HF) ordering
2. unschedule jobs that complete after $3LB$ (LB is lower bound on the global makespan), sort them by HF
3. schedule large ($> m/2$) jobs backwards from $3LB$
4. schedule remaining jobs in the gaps of the schedule

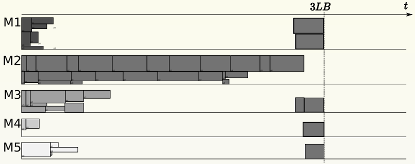
Example run: first, we ensure the worst-case performance . . .



(a) local scheduling

(b) MOCCA with gaps

Example run: . . . then, we collapse the schedule.



(b) MOCCA with gaps

(c) MOCCA, collapsed

Outline of the scheduling algorithm (MOCCA)

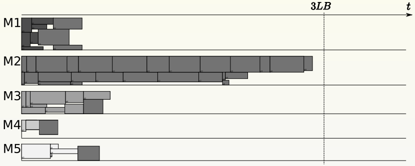
3-approximation of the global makespan; local constraints are not violated

1. schedule jobs locally using highest-first (HF)[1] ordering
2. unschedule jobs that complete after $3LB$ (LB is lower bound on the global makespan), sort them by HF
3. schedule large jobs ($> m/2$) backwards from $3LB$
4. schedule remaining jobs in the gaps of the schedule

[1] it is the natural extension of LPT...

Final load balancing improves organizations' makespans

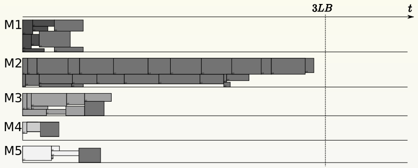
To improve (almost) everyone, we balance loads in order of increasing organizations' makespans.



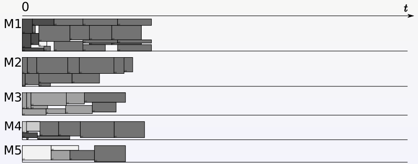
(c) collapsed schedule

Final load balancing improves organizations' makespans

To improve (almost) everyone, we balance loads in order of increasing organizations' makespans.



(e) collapsed schedule



(f) final load balancing

Summary: optimize the system goal, respect local goals

- Multi-Organization Scheduling Problem (MOSP): organizations have supercomputers and local jobs
- MOCCA does not worsen goals of organizations (local $C_{max}^{(u)}$)
- MOCCA 3-approximates the global makespan
- the organizations can not modify the proposed schedule

Outline

Links between local versus Global

Strict constraints

MOSP's local constraints (and also constraints like selfishness) are too strict in practice. They strongly limit the freedom of the scheduler to find a good global $C_{\max}$.

A clear trade-off

There is a correlation between the guarantees that we can provide individually for each organization and the global performance of the platform.

Question

How much can we improve the global $C_{\max}$ of the entire platform if we allow some controlled degradation of the local performance?

Links between local versus Global

Strict constraints

MOSP's local constraints (and also constraints like selfishness) are too strict in practice. They strongly limit the freedom of the scheduler to find a good global $C_{\max}$.

A clear trade-off

There is a correlation between the guarantees that we can provide individually for each organization and the global performance of the platform.

Question

How much can we improve the global $C_{\max}$ of the entire platform if we allow some controlled degradation of the local performance?

Links between local versus Global

Strict constraints

MOSP's local constraints (and also constraints like selfishness) are too strict in practice. They strongly limit the freedom of the scheduler to find a good global $C_{\max}$.

A clear trade-off

There is a correlation between the guarantees that we can provide individually for each organization and the global performance of the platform.

Question

How much can we improve the global $C_{\max}$ of the entire platform if we allow some controlled degradation of the local performance?

What is Fairness?

C_{max} is probably not the right objective (no meaning for the fairness).
Starting by a small (easy) example: two users are submitting their jobs, aiming each at minimizing the makespan of their jobs.
Let consider user 1 submits 2 jobs (4,4), same for user 2 who is submitting (3,7).

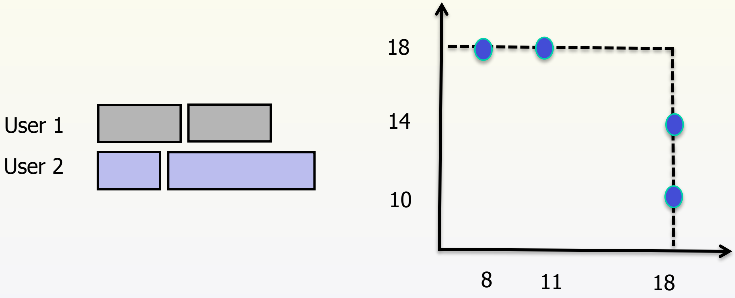
Question:
How many possible situations?

Pareto Optimality

Starting by a small (easy) example: two users are submitting their jobs, aiming each at minimizing the makespan of their jobs.
Let consider user 1 submits 2 jobs (4,4), same for user 2 who is submitting (3,7).

Question:
How many possible situations?

What is the best solution for each user?



Toward looking at Fairness in Combinatorial Optimization

The **stretch** (or slowdown factor) of job *i* is defined as: $s_i = \frac{C_i - r_i}{p_i}$.
Bounded stretch: $s_i = \frac{C_i - r_i}{max(\alpha, p_i)}$.

Question:
What are the (expected) results for max stretch and (weighted) sum stretch?

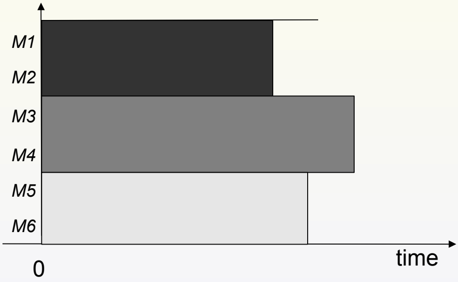
Toward looking at Fairness in Combinatorial Optimization

The **stretch** (or slowdown factor) of job *i* is defined as: $s_i = \frac{C_i - r_i}{p_i}$.
Bounded stretch: $s_i = \frac{C_i - r_i}{max(\alpha, p_i)}$.

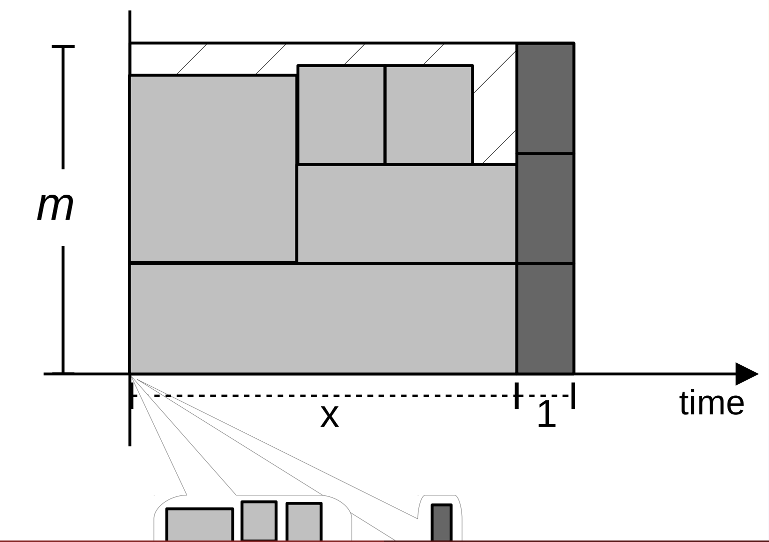
Question:
What are the (expected) results for max stretch and (weighted) sum stretch?

Adaptation to the campaign scheduling problem.

Classical fairsharing



Efficiency of FCFS



Outline

Centralized vs distributed

- Most efficient algorithms are centralized: they require global knowledge and a single executing entity.
- Scheduling (allocation) might become a bottleneck when systems are scaled to millions of cores.
- The answer: distributed multiobjective scheduling algorithms!
- Add fairness issues

Take home message

- Cooperation matters!
- Depending on the system, cooperation can be modelled using various techniques: optimization, multi-objective optimization, game theory (selfishness, fairness).
- We demonstrated how scheduling algorithms can be tuned to collaborative systems.