Non-Cooperative Scheduling of Multiple Bag-of-Tasks Applications

Infocom 2007

Arnaud Legrand, Corinne Touati

CNRS/INRIA, LIG laboratory

May 8th, 2007

Motivation

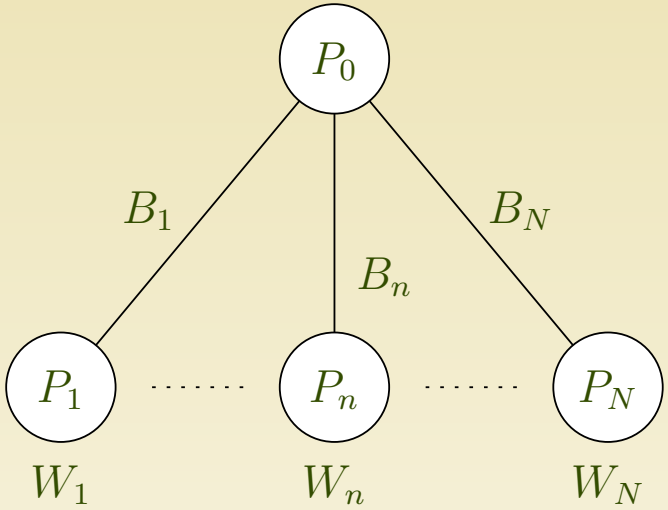
Large-scale distributed platforms result from the collaboration of many users:

- Multiple applications execute concurrently on heterogeneous platforms and compete for CPU and network resources.
- Sharing resources amongst users should somehow be fair. In a grid context, this sharing is generally done in the "low" layers (network, OS).
- We analyze the behavior of K non-cooperative schedulers that use the optimal strategy to maximize their own utility while fair sharing is ensured at a system level ignoring applications characteristics.

Outline

Outline

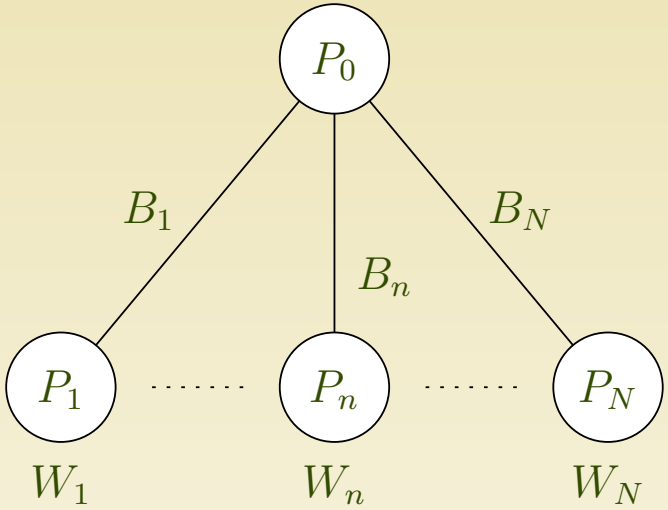
Master-Worker Platform



- N processors with processing capabilities W_n (in Mflop.s^{-1})
- using links with capacity B_n (in Mb.s^{-1})

Hypotheses :

Master-Worker Platform



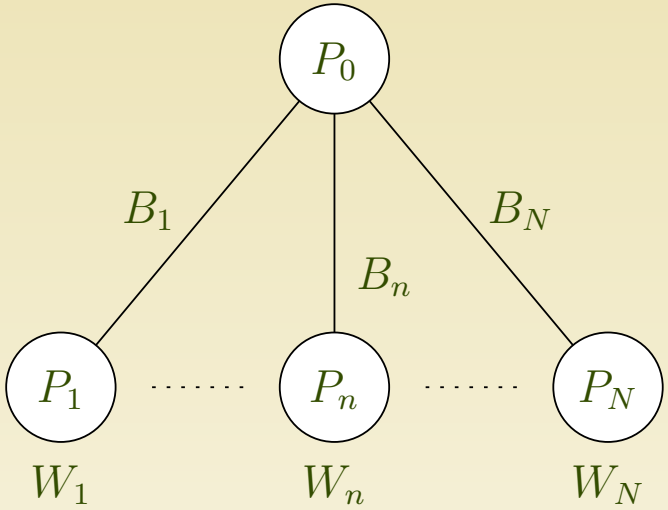
- N processors with processing capabilities W_n (in Mflop.s^{-1})
- using links with capacity B_n (in Mb.s^{-1})

Hypotheses :

- Multi-port

Communications to P_i do not interfere with communications to P_j.

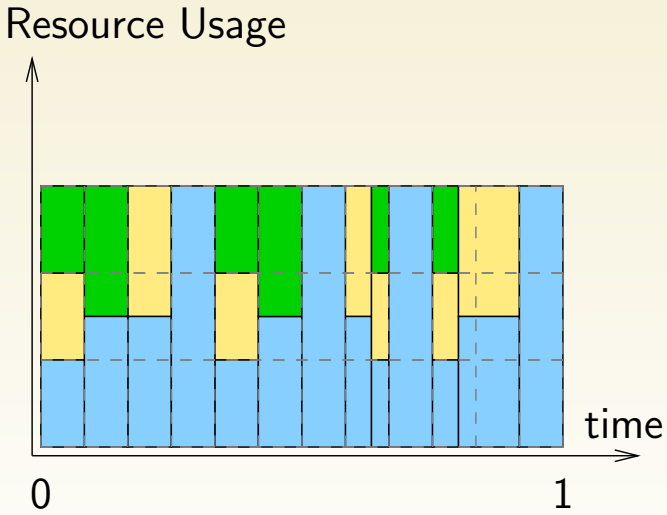
Master-Worker Platform



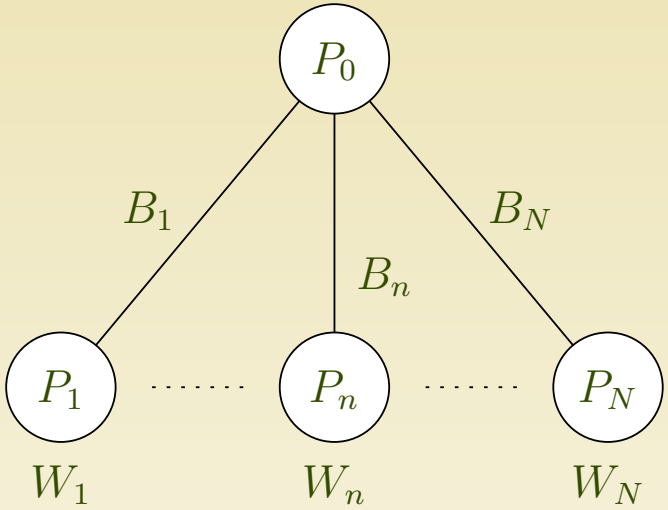
- N processors with processing capabilities W_n (in Mflop.s^{-1})
- using links with capacity B_n (in Mb.s^{-1})

Hypotheses :

- Multi-port
- No admission policy but an ideal local fair sharing of resources among the various requests



Master-Worker Platform



- N processors with processing capabilities W_n (in Mflop.s^{-1})
- using links with capacity B_n (in Mb.s^{-1})

Hypotheses :

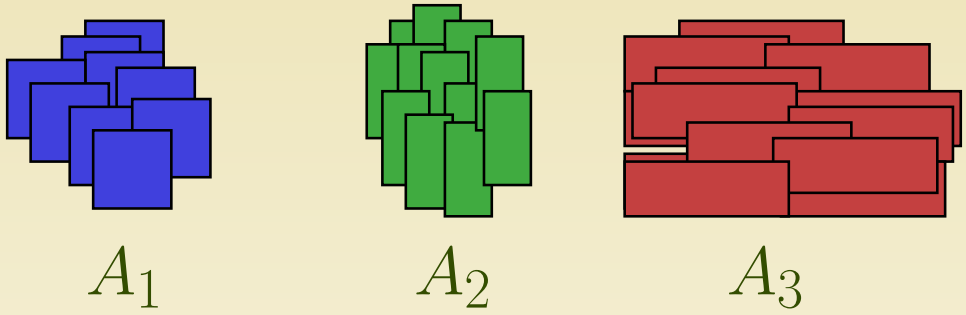
- Multi-port
- requests

Definition.

We denote by physical-system a triplet (N, B, W) where N is the number of machines, and B and W the vectors of size N containing the link capacities and the computational powers of the machines.

Applications

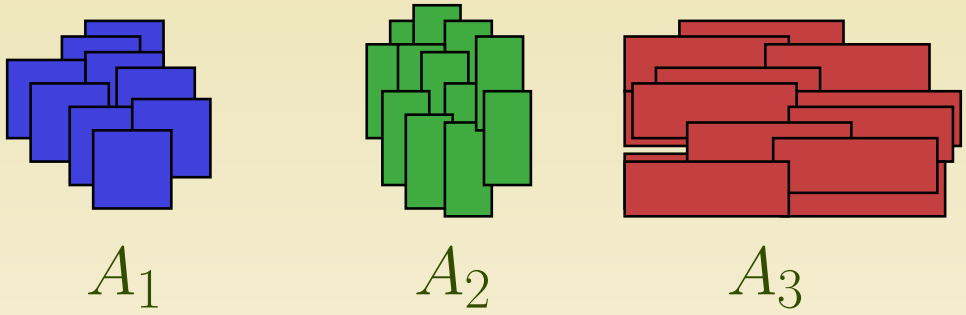
- Multiple applications ($A_1, \ldots, A_K$):



  - each consisting in a large number of same-size independent tasks
  - Different communication and computation demands for different applications. For each task of A_k:
    - processing cost w_k (MFlops)
    - communication cost b_k (MBytes)

Applications

- Multiple applications ($A_1, \ldots, A_K$):

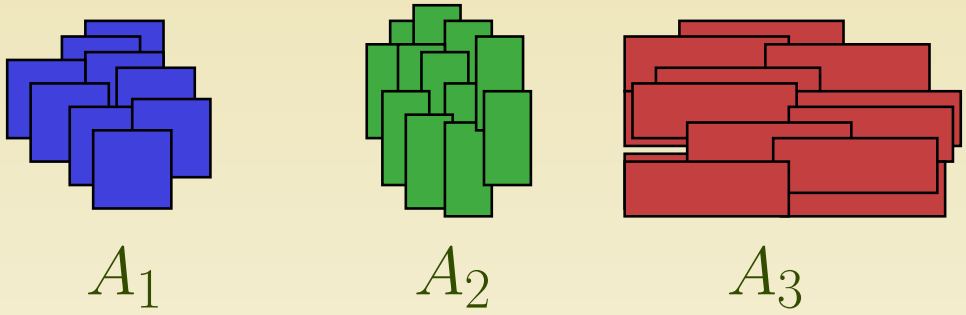


  - each consisting in a large number of same-size independent tasks
  - Different communication and computation demands for different applications. For each task of A_k:
    - processing cost w_k (MFlops)
    - communication cost b_k (MBytes)

- Master holds all tasks initially, communication for input data only (no result message).

Applications

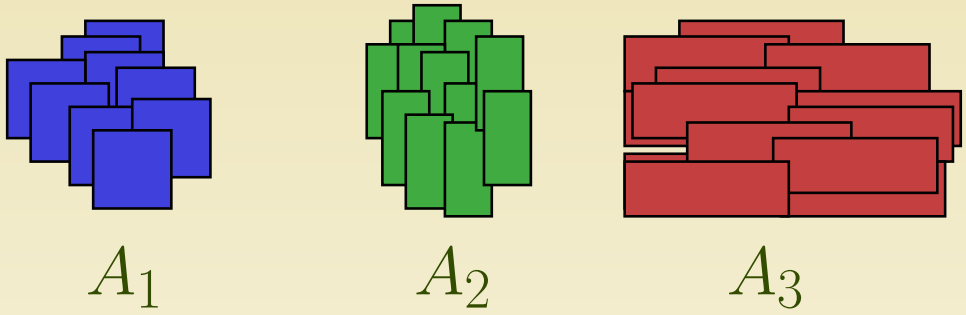
- Multiple applications ($A_1, \ldots, A_K$):



  - each consisting in a large number of same-size independent tasks
  - Different communication and computation demands for different applications. For each task of A_k:
    - processing cost w_k (MFlops)
    - communication cost b_k (MBytes)
- Master holds all tasks initially, communication for input data only (no result message).
- Such applications are typical desktop grid applications (SETI@home, Einstein@Home, ...)

Applications

- Multiple applications $(A_1, \ldots, A_K)$:



  - each consisting in a large number of same-size independent tasks
  - Different communication and computation demands for different applications. For each task of A_k:
    - processing cost w_k (MFlops)
    - communication cost b_k (MBytes)

Definition.

We define an application-system as a triplet (K, b, w) where K is the number of applications, and b and w the vectors of size K representing the size and the amount of computation associated to the different applications.

Steady-State scheduling

In the following our K applications run on our N workers and compete for network and CPU access:

Definition.

A system S is a sextuplet (K, b, w, N, B, W), with K,b,w,N,B,W defined as for a user-system and a physical-system.

- Task regularity $\rightsquigarrow$ steady-state scheduling.
- Maximize throughput (average number of tasks processed per unit of time)

$$\alpha_k = \lim_{t \to \infty} \frac{done_k(t)}{t}.$$

Similarly: $\alpha_{n,k}$ is the average number of tasks of type k performed per time-unit on the processor P_n.

$$\alpha_k = \sum_n \alpha_{n,k}.$$

- α_k is the utility of application k.

Constraints

The scheduler of each application thus aims at maximizing its own throughput, i.e. α_k.

However, as applications use the same set of resources, we have the following general constraints:

Computation $\forall n \in [\![0, N]\!] : \sum_{k=1}^{K} \alpha_{n,k} \cdot w_k \leqslant W_n$

Communication $\forall n \in [\![1, N]\!] : \sum_{k=1}^{K} \alpha_{n,k} \cdot b_k \leqslant B_n$

Applications should decide when to send data from the master to a worker and when to use a worker for computation.

Outline

Optimal strategy for a single application

Single application

This problem reduces to maximizing $\sum_{n=1}^{N} \alpha_{n,1}$ while:

$$\begin{cases} \forall n \in [\![A, N]\!] : \alpha_{n,1} \cdot w_1 \leqslant W_n \\ \forall n \in [\![1, N]\!] : \alpha_{n,1} \cdot b_1 \leqslant B_n \\ \forall n, \quad \alpha_{n,1} \geqslant 0. \end{cases}$$

The optimal solution to this linear program is obtained by setting

$$\forall n,\ \alpha_{n,1} = \min\left(\frac{W_n}{w_1}, \frac{B_n}{b_1}\right)$$

In other words

The master process should saturate each worker by sending it as many tasks as possible.
A simple acknowledgment mechanism enables the master process to ensure that it is not over-flooding the workers, while always converging to the optimal throughput.

A Non-Cooperative Game

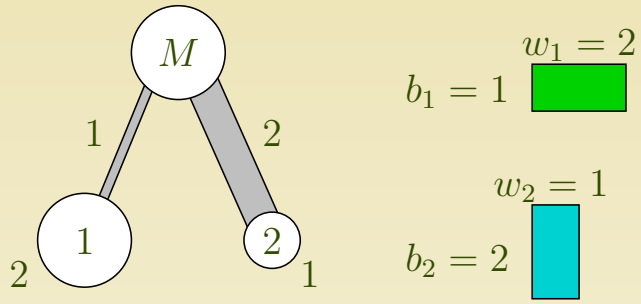
We suppose a purely non-cooperative game where no scheduler decides to "ally" to any other (i.e. no coalition is formed).

As the players constantly adapt to each others' actions, they may (or not) reach some equilibrium, known in game theory as Nash equilibrium [Nas50, Nas51].

$\alpha_{n,k}^{(nc)}$ denotes the rates achieved at such stable states.

A simple example

Two computers 1 and 2: $B_1 = 1$, $W_1 = 2$, $B_2 = 2$, $W_2 = 1$.
Two applications: $b_1 = 1$, $w_1 = 2$, $b_2 = 2$ and $w_2 = 1$.

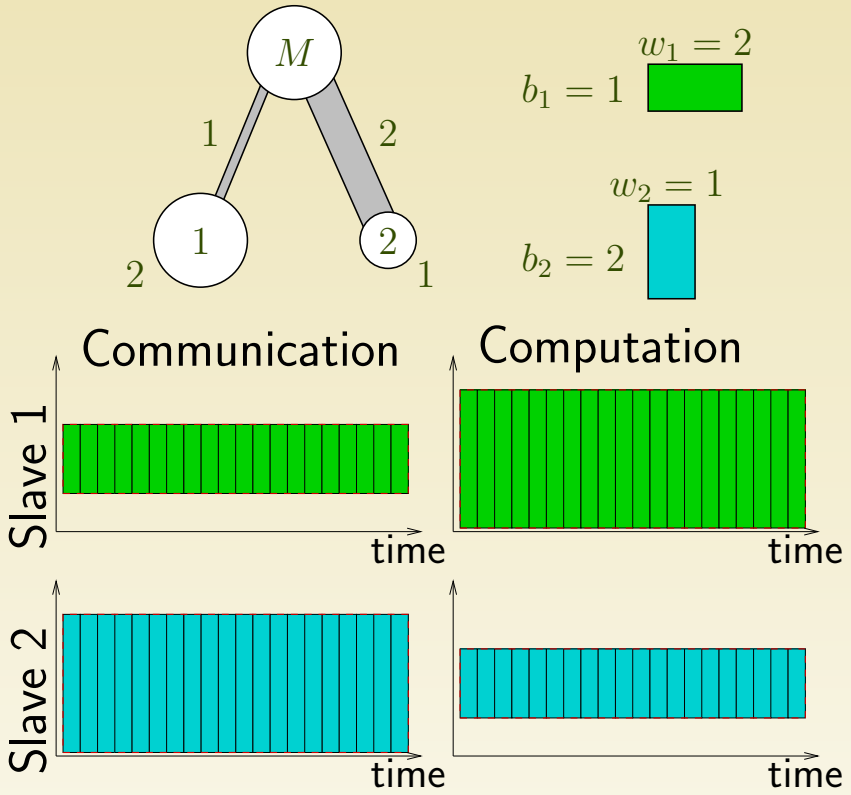


A simple example

Two computers 1 and 2: $B_1 = 1$, $W_1 = 2$, $B_2 = 2$, $W_2 = 1$.
Two applications: $b_1 = 1$, $w_1 = 2$, $b_2 = 2$ and $w_2 = 1$.

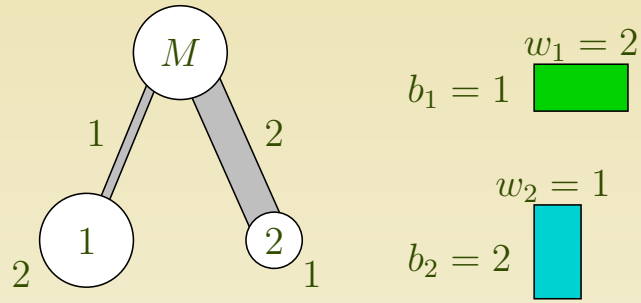
Cooperative Approach:

Application 1 is processed exclusively on computer 1 and application 2 on computer 2.
The respective throughput is $\alpha_1^{(\text{coop})} = \alpha_2^{(\text{coop})} = 1$.



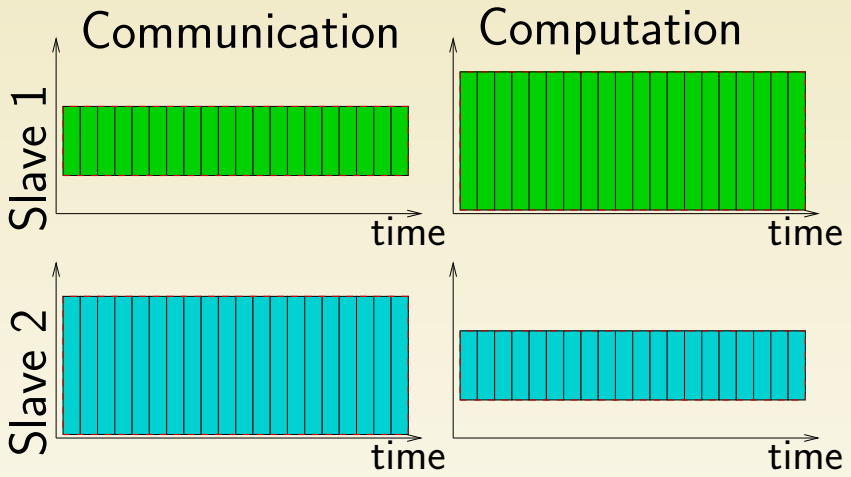
A simple example

Two computers 1 and 2: $B_1 = 1$, $W_1 = 2$, $B_2 = 2$, $W_2 = 1$.
Two applications: $b_1 = 1$, $w_1 = 2$, $b_2 = 2$ and $w_2 = 1$.



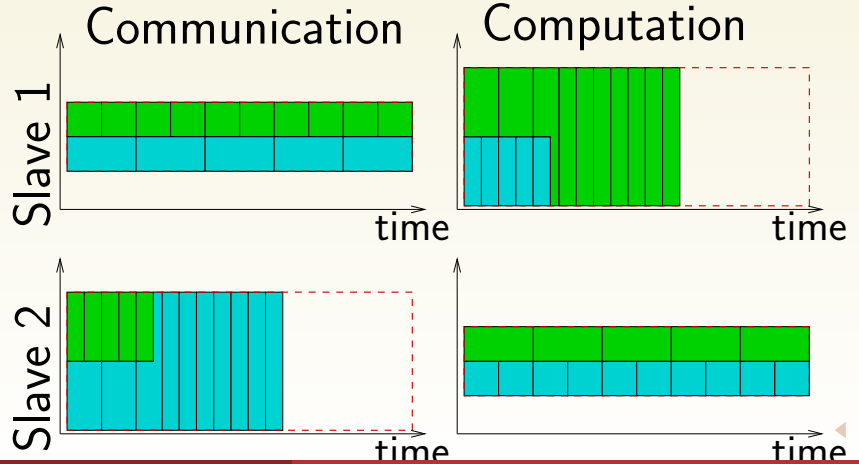
Cooperative Approach:

Application 1 is processed exclusively on computer 1 and application 2 on computer 2.
The respective throughput is $\alpha_1^{(\text{coop})} = \alpha_2^{(\text{coop})} = 1$.



Non-Cooperative Approach:

$\alpha_1^{(nc)} = \alpha_2^{(nc)} = \frac{3}{4}$



Characterizing the Nash Equilibrium

Theorem 1.

For a given system (N, B, W, K, b, w) there exists exactly one Nash Equilibrium and it can be analitycally computed.

Proof.

Under the non-cooperative assumption, on a given worker, an application is either communication-saturated or computation-saturated.

Puting schedules in some canonical form enables to determine for each processor, which applications are communication-saturated and which ones are computation-saturated and then to derive the corresponding rates. □

Outline

Pareto Optimality

Definition: Pareto optimality.

An allocation is said to be Pareto-optimal if it is impossible to strictly increase the throughput of an application without strictly decreasing the one of another.

When is our Nash Equilibrium Pareto-optimal ?

Pareto Optimality

Definition: Pareto optimality.

An allocation is said to be Pareto-optimal if it is impossible to strictly increase the throughput of an application without strictly decreasing the one of another.

When is our Nash Equilibrium Pareto-optimal ?

Theorem 2.

The allocation at the Nash equilibrium is Pareto inefficient if and only if there exists two workers, namely n_1 and n_2 such that all applications are communication-saturated on n_1 and computation-saturated on n_2 (i.e. $\sum_k \frac{B_{n_1}}{W_{n_1}} \frac{w_k}{b_k} \leqslant K$ and $\sum_k \frac{b_k}{w_k} \frac{W_{n_2}}{B_{n_2}} \leqslant K$).

Corrolary: on a single-processor system, the allocation at the Nash equilibrium is Pareto optimal.

Braess-like Paradox

Pareto-inefficient equilibria can exhibit unexpected behavior.

Definition: Braess Paradox.

There is a Braess Paradox if there exists two systems ini and aug such that

$$ini < aug \text{ and } \alpha^{(nc)}(ini) > \alpha^{(nc)}(aug).$$

Braess-like Paradox

Pareto-inefficient equilibria can exhibit unexpected behavior.

Definition: Braess Paradox.

There is a Braess Paradox if there exists two systems ini and aug such that

$$ini < aug \text{ and } \alpha^{(nc)}(ini) > \alpha^{(nc)}(aug).$$

Theorem 3.

In the non-cooperative multi-port scheduling problem, Braess like paradoxes cannot occur.

Proof.

- Defining an equivalence relation on sub-systems.
- Defining an order relation on equivalent sub-systems.

□

Definition [KP98]

$$\phi_\Sigma = \max_S \frac{\sum_k \alpha_k^{(\Sigma)}(S)}{\sum_k \alpha_k^{(nc)}(S)} \geqslant 1.$$

Definition [KP98] More generally, one can define:

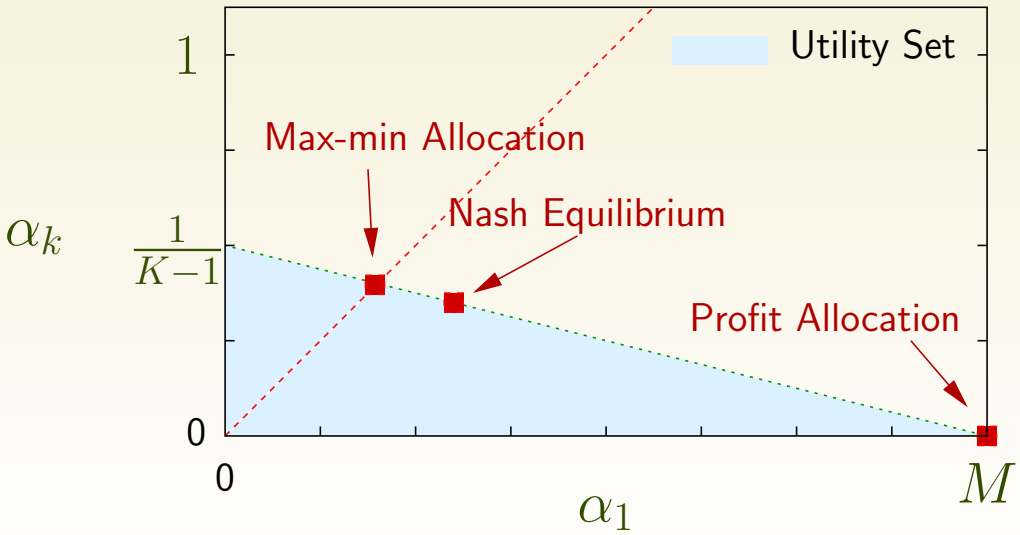
$$I_f(S) = \frac{f\left(\alpha_1^{(f)}(S), \ldots, \alpha_K^{(f)}(S)\right)}{f\left((\alpha_1^{(nc)}(S), \ldots, \alpha_K^{(nc)}(S)\right)} \geqslant 1.$$

Definition [KP98] More generally, one can define:

$$I_f(S) = \frac{f\left(\alpha_1^{(f)}(S), \ldots, \alpha_K^{(f)}(S)\right)}{f\left((\alpha_1^{(nc)}(S), \ldots, \alpha_K^{(nc)}(S)\right)} \geqslant 1.$$

Problem measures the "distance" to a particular point...

Illustration 1 machine ($B_1 = W_1 = 1$) and K applications $b = (\frac{1}{M}, 1...1)$ and $w = (\frac{1}{M}, 1...1)$.



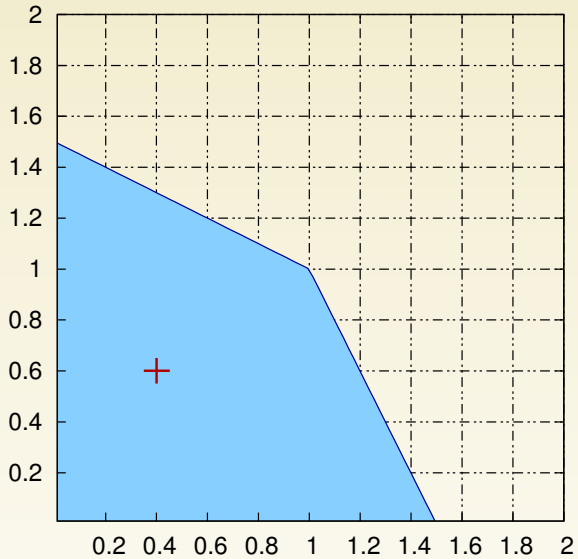
$$I_\Sigma(S_{M,K}) = \frac{M}{\frac{M}{K} + \frac{K-1}{K}} \xrightarrow[M \to \infty]{} K$$

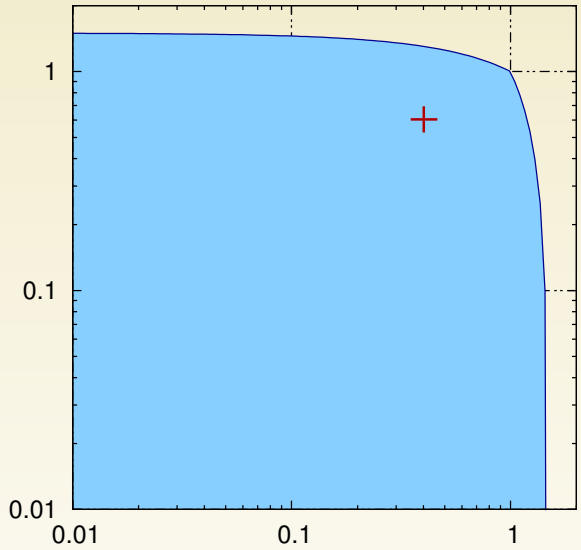
Utility set and allocations $S_{M,K}$ ($K = 3, M = 2$).

Rationale measure the "distance" to the Pareto set.

$$\phi = \max_S I(S) = \max_S \max_{\alpha \in U(S)} \min_k \frac{\alpha_k}{\alpha_{n,k}^{(nc)}(S)}.$$

Illustration





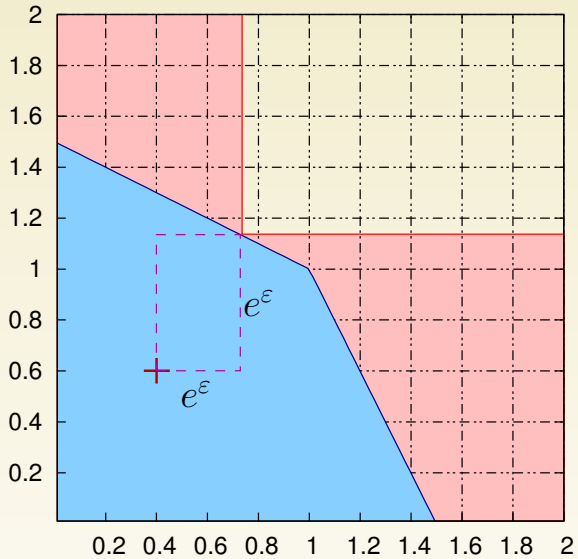
Degradation Factor is related to ε-approximation of Pareto-curves [PY00].

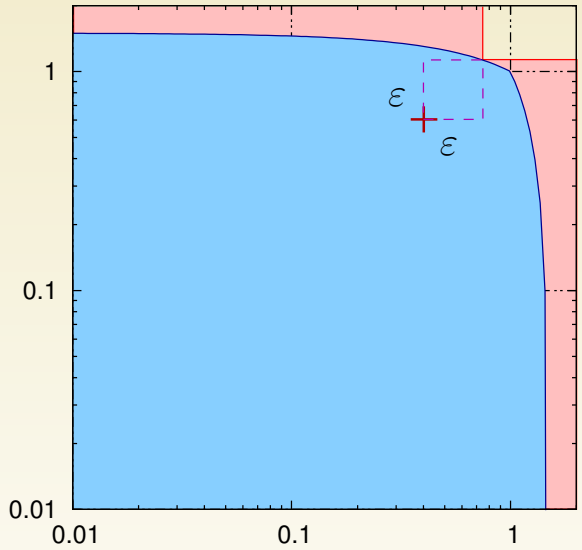
Here Selfishness Degradation Factor is at least 2.

Rationale measure the "distance" to the Pareto set.

$$\phi = \max_S I(S) = \max_S \max_{\alpha \in U(S)} \min_k \frac{\alpha_k}{\alpha_{n,k}^{(nc)}(S)}.$$

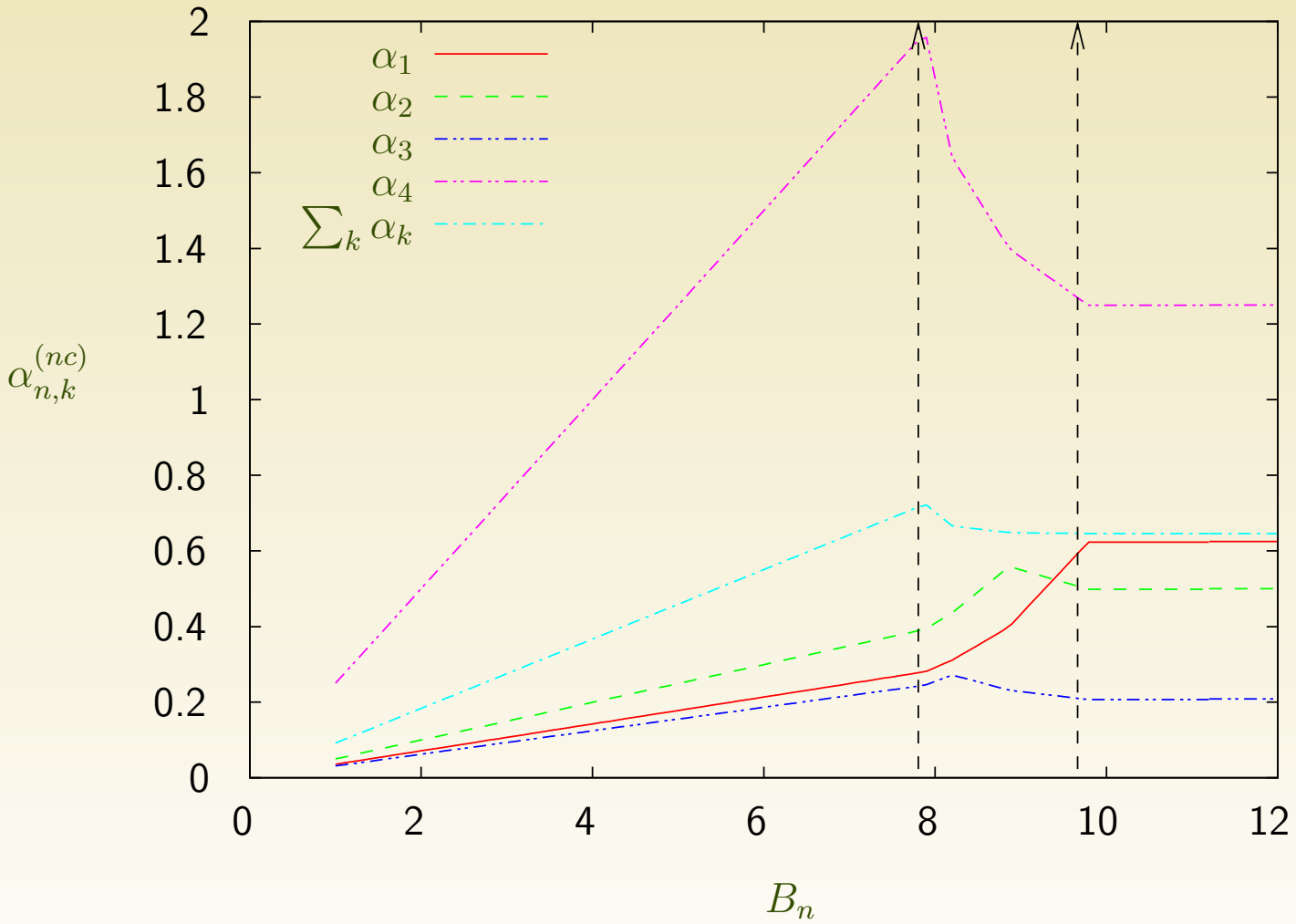
Illustration





Degradation Factor is related to ε-approximation of Pareto-curves [PY00].
Here Selfishness Degradation Factor is at least 2.

Pareto optimality and monotonicity of performance measures



Most classical performance measures decrease with resource augmentation!

Outline

Conclusion

- Applying fair and optimal sharing on each resource does not ensure any fairness nor efficiency when users do not cooperate.

  $\rightsquigarrow$ either applications cooperate or new complex and global access policies should be designed

Future Work

- Measuring Pareto-inefficiency is an open question that we are currently investigating.
- In the one-port communication model, Braess-like paradoxes seem to arise.

Bibliography


E. Koutsoupias and C. Papadimitriou.
Worst-case equilibria.
In *STACS*, 1998.


John F. Nash.
Equilibrium points in n-person games.
Proceedings of the National Academy of Sciences USA, 36:48–49, 1950.


John F. Nash.
Noncooperative games.
Annal of Mathematics, 54:286–295, 1951.


C. H. Papadimitriou and M. Yannakakis.
On the approximability of trade-offs and optimal access of web sources.
In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 86, Washington, DC, USA, 2000. IEEE Computer Society.