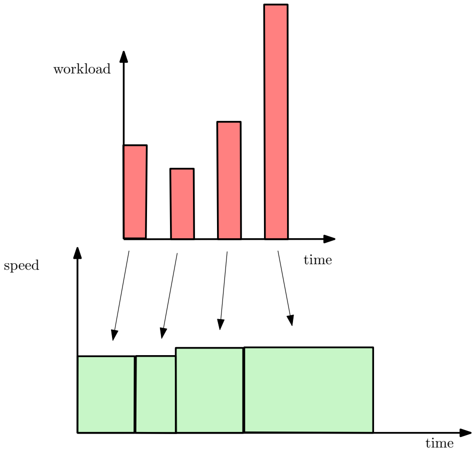
Learning Augmented Energy Minimization via Speed Scaling

Etienne Bamas, Andreas Maggiori, Lars Rohwedder, Ola Svensson

EPFL

NeurIPS 2020

Problem



- Every millisecond, the server receives a job to execute.
- Each job comes with some workload w_i that must be finished within D milliseconds after arrival.
- The server can choose its processor's speed $s(t)$ at will.
- The goal is to minimize the energy

$$\int s(t)^{\alpha} dt$$

for a fixed $\alpha > 1$.

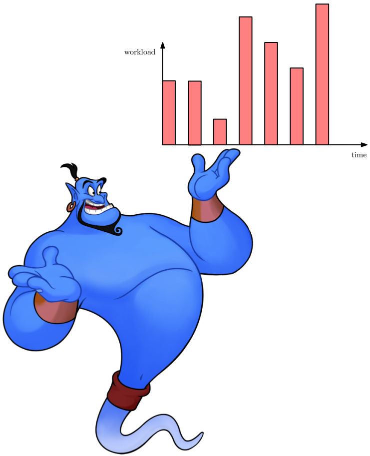
Prior work

- Introduced by Yao et al. (FOCS 1995).
- Greedy algorithm (Yao et al.) solves the offline problem optimally.

- **Online** problem: w_i is revealed at time i not before! This problem is well understood.
- Average Rate algorithm is $2^{\alpha-1}\alpha^{\alpha}$-competitive (Yao et al. FOCS' 95).
- Optimal Available is α^{α}-competitive (Bansal et al. J. ACM 2007).
- BKP is $O(e^{\alpha})$-competitive (Bansal et al. J. ACM 2007)

- The competitive ratio has to be **exponential** in α.

What if we could imperfectly see the future?

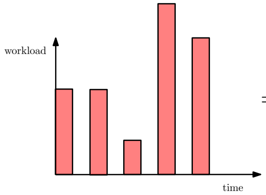


Our new problem: design an algorithm that outperforms any purely online algorithm if err is small and stays comparable to online algorithms when err is big.

This is referred to as **learning augmented** algorithms. A recent but quickly growing line of work:

- Competitive caching (Lykouris and Vassilvitskii ICML 2018)
- Ski rental (Kumar et al. NeurIPS 2018, Gollapudi ICML 2019)
- Scheduling (Lattanzi et al. SODA 2020)
- Frequency estimation (Hsu et al. ICLR 2019)

How do we define the error err?

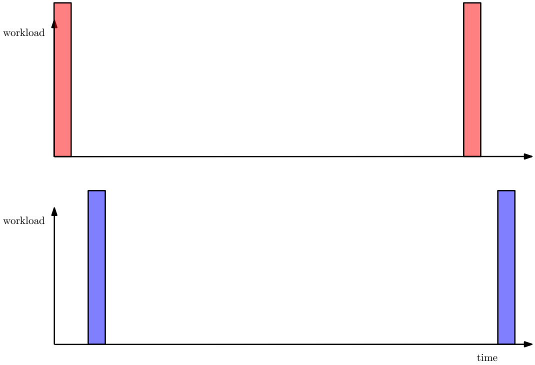


$= (5, 5, 2, 10, 8, \ldots)$

- Instance can be seen as a workload vector.
- What metric to use to compare w^{pred} and w^{real}?
- Simplest metrics $||.||_1$, $||.||_2$ do not give enough information!
- We will define

$$\text{err} = ||w^{\text{pred}} - w^{\text{real}}||_\alpha^\alpha$$

But is $||.||_\alpha^\alpha$ a good metric?



- We show how to make this metric much more robust to small shifts in the timeline

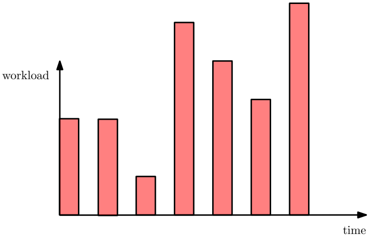
A first learning augmented algorithm

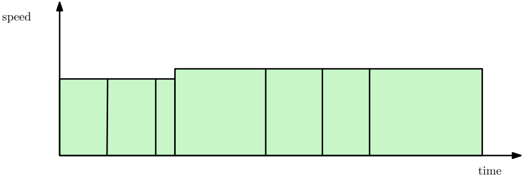
- $\mathrm{err} = ||w^{\mathrm{pred}} - w^{\mathrm{real}}||_\alpha^\alpha = \sum_{i \geq 0} |w^{\mathrm{pred}} - w^{\mathrm{real}}|^\alpha$

- If $\mathrm{err} \approx 0$ (i.e. the prediction is very good), the algorithm should be **much better** than an online algorithm without prediction. We say it is **consistent**.

- No matter how big err is, the algorithm should **always** be competitive against offline OPT (comparable to what an online algorithm without prediction would give). We say it is **robust**.

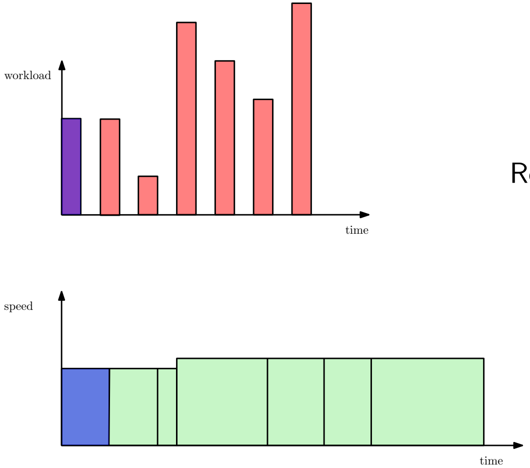
A first learning augmented algorithm





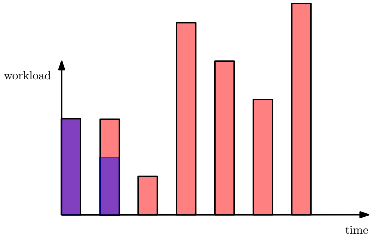
Compute an optimum schedule for the prediction.

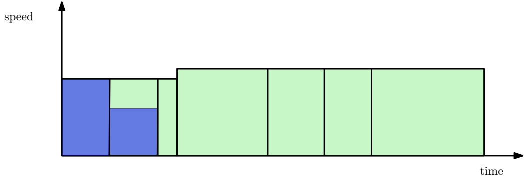
A first learning augmented algorithm



Receive the real instance online.

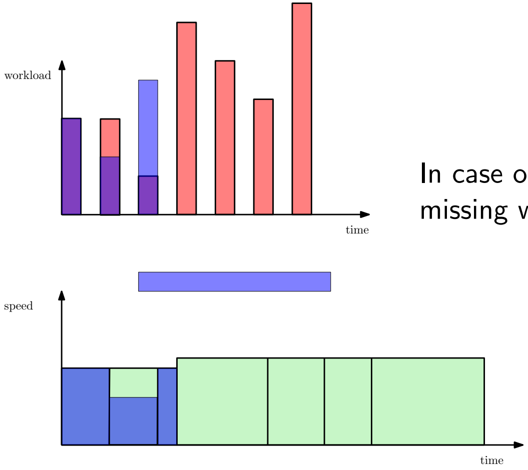
A first learning augmented algorithm





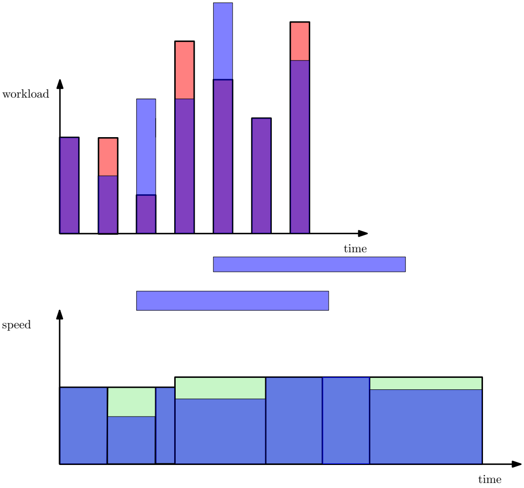
In case of over prediction, scale down the speed.

A first learning augmented algorithm



In case of under prediction for job i, spread uniformly the missing work in the interval $[i, i + D]$.

A first learning augmented algorithm



What guarantees for this algorithm?

- The cost is **always** at most

$$(1+\epsilon)\mathrm{OPT} + O\left(\frac{\alpha}{\epsilon}\right)^{\alpha} \mathrm{err}$$

for any $\epsilon > 0$.

- Is it **robust**? No! An arbitrarily bad prediction can lead to arbitrarily bad performance of this algorithm.

What guarantees for this algorithm?

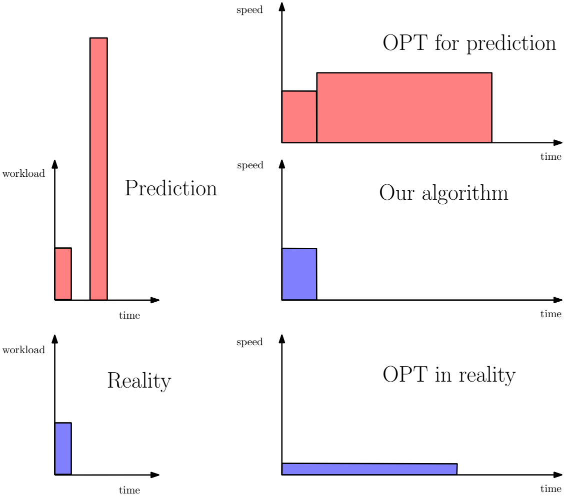
- The cost is **always** at most

$$(1+\epsilon)\mathrm{OPT} + O\left(\frac{\alpha}{\epsilon}\right)^{\alpha} \mathrm{err}$$

  for any $\epsilon > 0$.

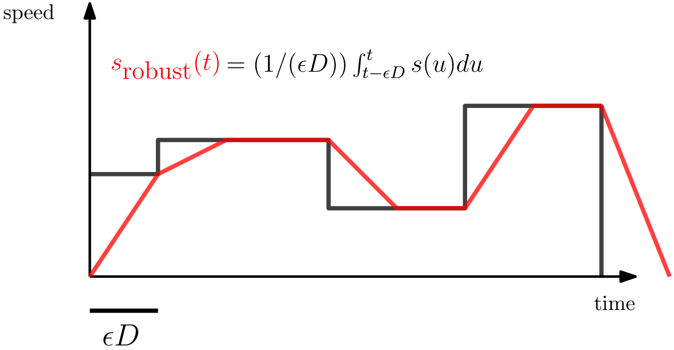
- Is it **robust**? No! An arbitrarily bad prediction can lead to arbitrarily bad performance of this algorithm.

The bad example



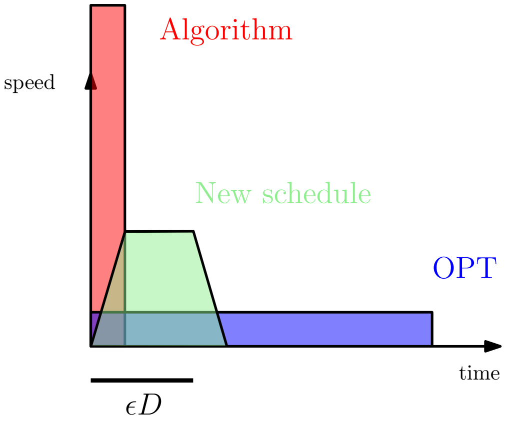
How to make an algorithm robust?

- **Idea:** Average out the speed to avoid huge peaks.



Why does it work?

Let's see the bad example again.



What about the deadlines?

- **Problem:** we are introducing a delay of ϵD in the schedule. Some deadlines might be not respected!

- **Fix:** Run the algorithm with shorter deadlines.

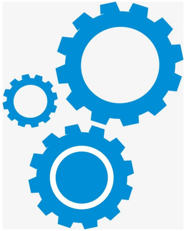
$$D \longleftarrow (1-\epsilon)D$$

- We show that this increases the cost of OPT only by a multiplicative factor $\approx (1+\epsilon)^{\alpha-1}$.

Summary of this method

Given any algorithm A that outputs a feasible schedule we obtain a feasible schedule that is also robust!

Convolution



$$\text{Schedule of cost } c(A) \longrightarrow \text{[Convolution]} \longrightarrow \min\{(1+\epsilon)c(A) \quad , \quad O\left(\frac{\alpha}{\epsilon}\right)^{\alpha} \mathrm{OPT}\}$$

Summary of our results

- We design an algorithm that outputs a feasible schedule whose cost can be bounded as follows for any $\epsilon > 0$.

Prediction

good   bad

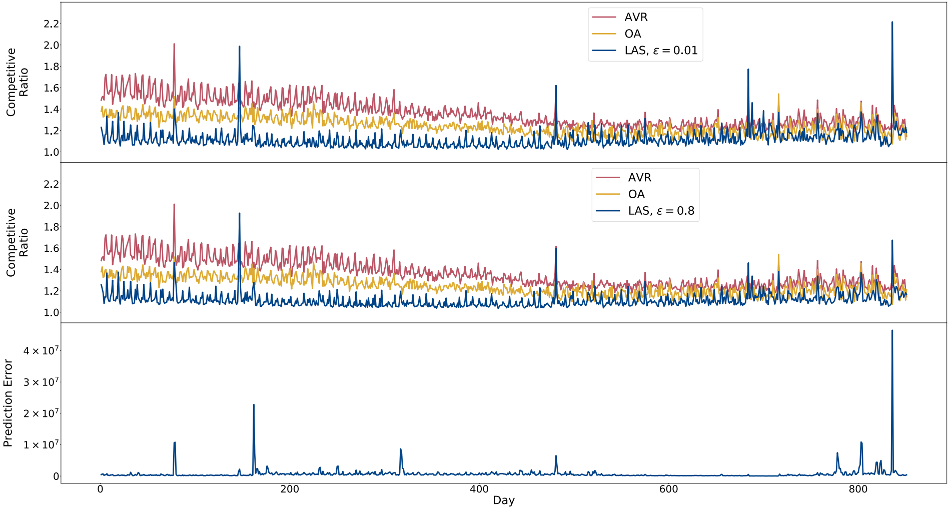
$$\text{Cost of the schedule} \quad \leq \quad \min \left\{ \underbrace{(1+\epsilon)\mathrm{OPT} + O\left(\frac{\alpha}{\epsilon}\right)^{\alpha} \mathrm{err}}_{\text{good}} \quad , \quad \underbrace{O\left(\frac{\alpha}{\epsilon}\right)^{\alpha} \mathrm{OPT}}_{\text{bad}} \right\}$$

Additional results

- We give an algorithm with a similar guarantee with a respect to a more robust notion of error (allowing to shift the timeline).
- We get similar results in the case of general deadlines (not all jobs have the same time D to be completed). In this case the convolution does not work!
- Experimental validation of our algorithm.

Experimental results

Results obtained with a very simple prediction!



Thank you for your attention.