# Learning to control large scale parallel computing platforms.

## Tuning Backfilling Queues

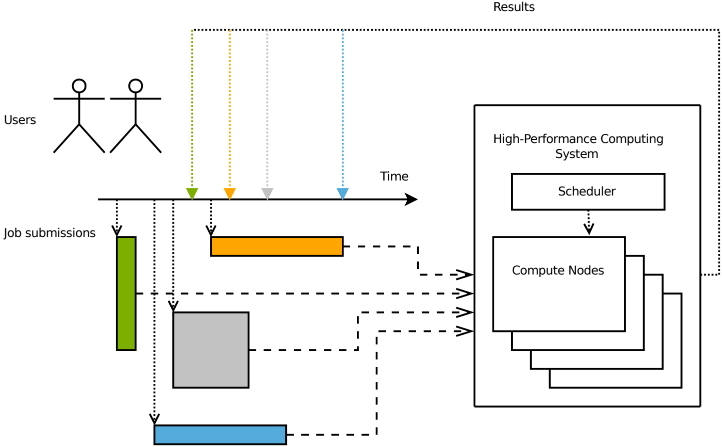
Valentin Reis
advised by Denis Trystram(LIG) and Jerôme Lelong(LJK)

Journées scientifiques PERSYVAL-Lab, June 13 2017

The investing institution/company sees this 10M cores machine:


It finds the initial 280M USD and sustains the 15MW peak power.

# The machine is used by submitting jobs.



## The system administrator sees this:

The users see this:

## Problem

**Find a policy for** the *on-line nonpreemptive execution of a set of parallel jobs on a HPC platform with a complex communication network linking heterogenous resources.*

## Problem

**Find a policy for** the *on-line nonpreemptive execution of a set of parallel jobs on a HPC platform with a complex communication network linking heterogenous resources.*

## Objective

Minimize the average waiting time of jobs.
*TODO: bus stop*

## Problem

**Find a policy for** the *on-line nonpreemptive execution of a set of parallel jobs on a HPC platform with a complex communication network linking heterogenous resources.*

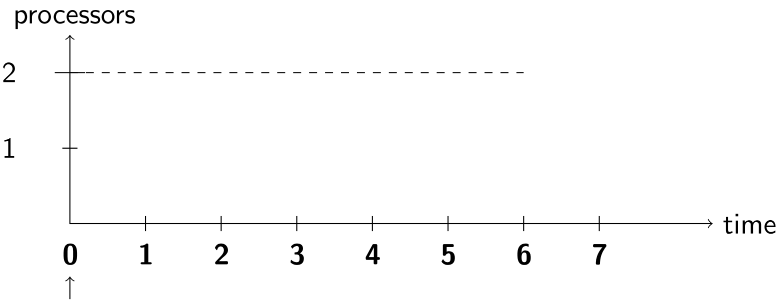
## Objective

Minimize the average waiting time of jobs.
*TODO: bus stop* **The elephant in the room**

The performance of any scheduling policy is **heavily dependent on user and job behavior.**
Our answer: **adaptation**.

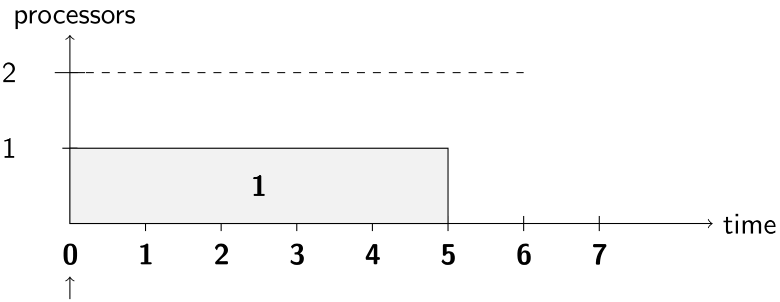
2 The current state of affairs
- Backfilling heuristics
- Tuning

## The basic heuristic: **EASY-Backfilling**



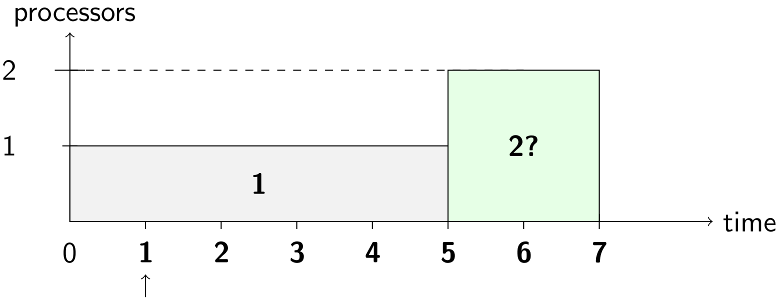

Submission dates
Resource requirements
Requested running times
Running time

# The basic heuristic: **EASY-Backfilling**



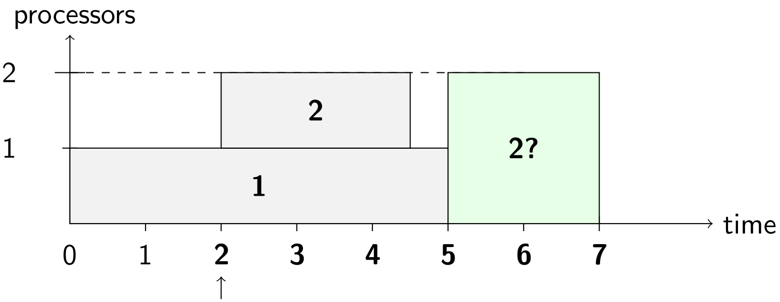

| | |
|---|---|
| Submission dates | $r_1 = 0$ |
| Resource requirements | $q_1 = 1$ |
| Requested running times | $\widetilde{p_1} = 5$ |
| Running time | |

## The basic heuristic: **EASY-Backfilling**



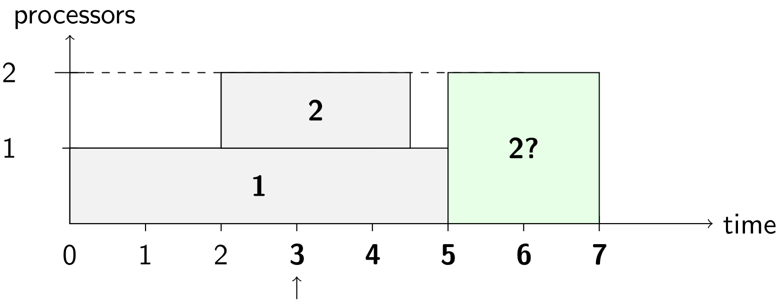

| | | |
|---|---|---|
| Submission dates | $r_1 = 0$ | $r_2 = 1$ |
| Resource requirements | $q_1 = 1$ | $q_2 = 2$ |
| Requested running times | $\widetilde{p_1} = 5$ | $\widetilde{p_2} = 2$ |
| Running time | | |

## The basic heuristic: **EASY-Backfilling**



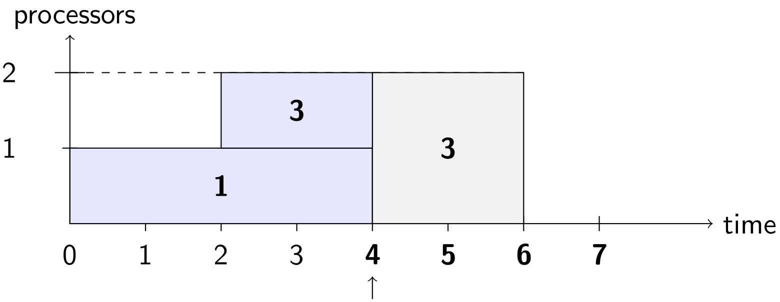

| | | | |
|---|---|---|---|
| Submission dates | $r_1 = 0$ | $r_2 = 1$ | $r_3 = 2$ |
| Resource requirements | $q_1 = 1$ | $q_2 = 2$ | $q_3 = 1$ |
| Requested running times | $\widetilde{p_1} = 5$ | $\widetilde{p_2} = 2$ | $\widetilde{p_3} = 2.5$ |
| Running time | | | |

## The basic heuristic: **EASY-Backfilling**



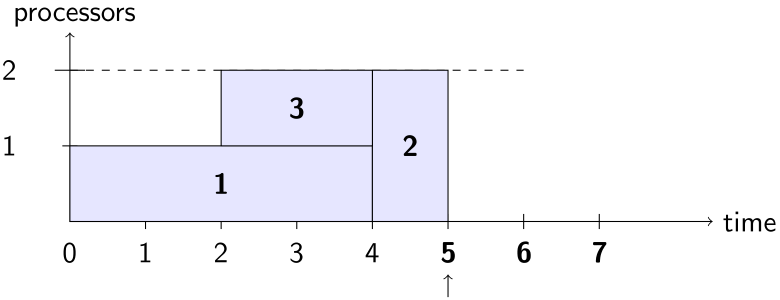

| | | | |
|---|---|---|---|
| Submission dates | $r_1 = 0$ | $r_2 = 1$ | $r_3 = 2$ |
| Resource requirements | $q_1 = 1$ | $q_2 = 2$ | $q_3 = 1$ |
| Requested running times | $\widetilde{p_1} = 5$ | $\widetilde{p_2} = 2$ | $\widetilde{p_3} = 2.5$ |
| Running time | | | |

# The basic heuristic: **EASY-Backfilling**




| | | | |
|---|---|---|---|
| Submission dates | $r_1 = 0$ | $r_2 = 1$ | $r_3 = 2$ |
| Resource requirements | $q_1 = 1$ | $q_2 = 2$ | $q_3 = 1$ |
| Requested running times | $\widetilde{p_1} = 5$ | $\widetilde{p_2} = 2$ | $\widetilde{p_3} = 2.5$ |
| Running time | $p_1 = 4$ | | $p_3 = 2$ |

## The basic heuristic: **EASY-Backfilling**




| | | | |
|---|---|---|---|
| Submission dates | $r_1 = 0$ | $r_2 = 1$ | $r_3 = 2$ |
| Resource requirements | $q_1 = 1$ | $q_2 = 2$ | $q_3 = 1$ |
| Requested running times | $\widetilde{p_1} = 5$ | $\widetilde{p_2} = 2$ | $\widetilde{p_3} = 2.5$ |
| Running time | $p_1 = 4$ | $p_2 = 1$ | $p_3 = 2$ |

# Primary and Backfilling Reordering Policies

The 'primary' and 'backfilling' job order may be independently tampered with. Many heuristics exist.

- FCFS: First-Come First-Serve, the **widely used default policy which ensures no starvation**
- LCFS: Last-Come First-Serve.
- LPF: Longest estimated Processing time First.
- SPF: Smallest estimated Processing time First.
- LQF: Largest resource requirement First.
- SQF: Smallest resource requirement First.
- EXP: Largest Expansion Factor First

Problem statement: **Can we leverage logged machine usage data in order to choose both primary and backfilling policy among the various available heuristics?**

3 Our approach
- Contributions
- Resampling methodology.
- Managing risk with thresholding.

Our contributions:

- A new lightweight HPC Simulator
- The study of static policies under a **resampling-based, train/test** methodology.
- How to avoid 'extreme waiting time' events?

# Resampling: why?

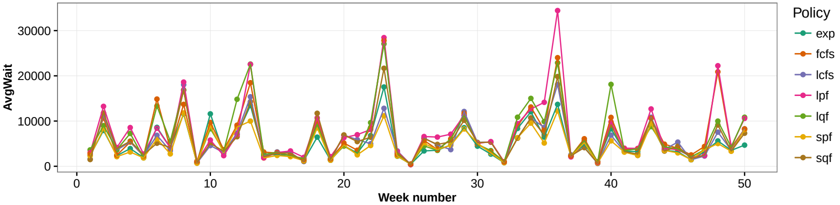
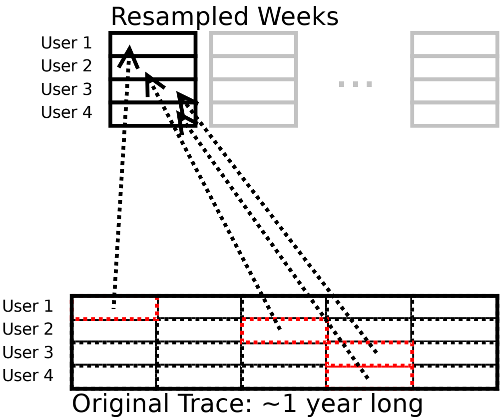


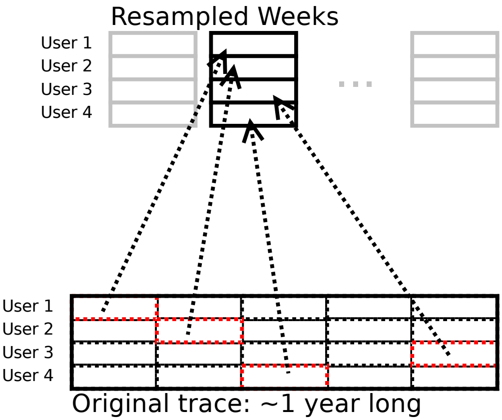

Figure: Weekly average waiting times of various policies.

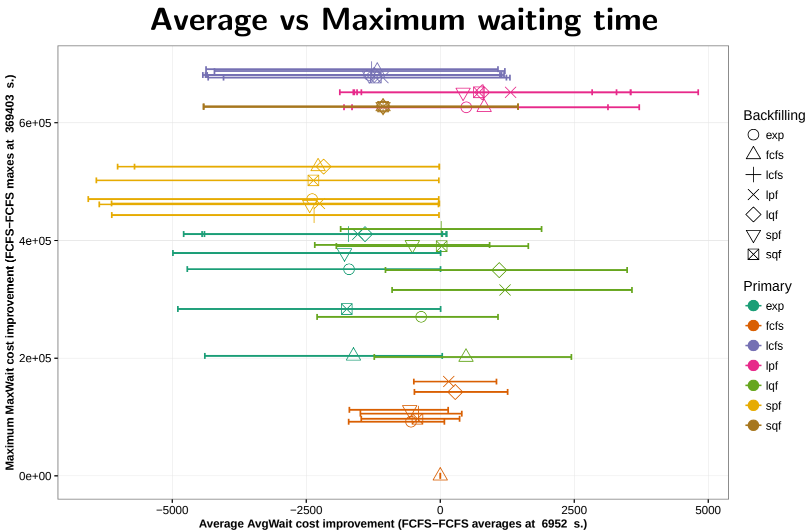
We need larger sample sizes.

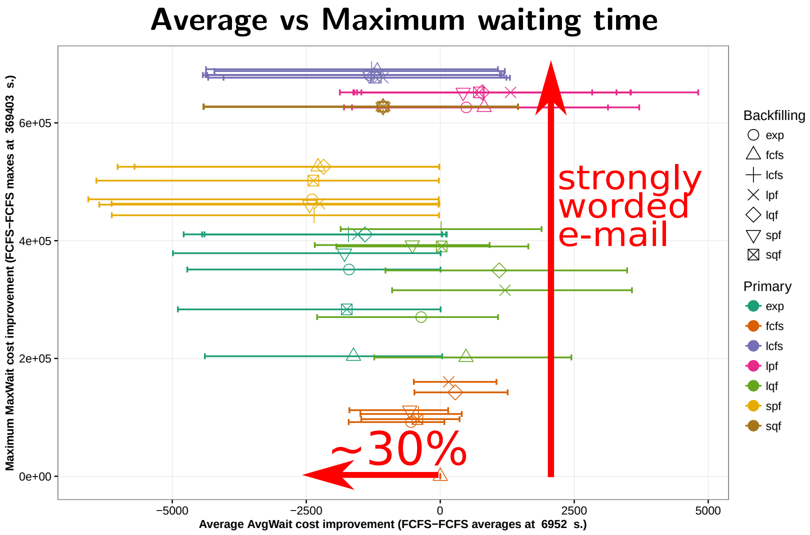
Resampling, or: **how to simulate using 2000 weeks of log data as input using a year-long trace.**



Resampling, or: how to simulate using 2000 weeks of log data as input using a year-long trace.



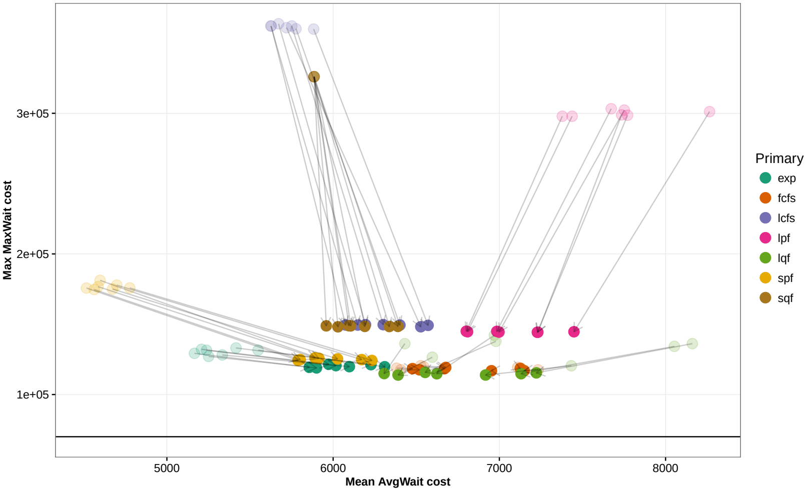

Average vs Maximum waiting time
Maximum MaxWait cost improvement (FCFS−FCFS maxes at 369403 s.)
0e+00
2e+05
4e+05
6e+05
−5000
−2500
0
2500
5000
Average AvgWait cost improvement (FCFS−FCFS averages at 6952 s.)
Backfilling
exp
fcfs
lcfs
lpf
lqf
spf
sqf
Primary
exp
fcfs
lcfs
lpf
lqf
spf
sqf


Average vs Maximum waiting time
Maximum MaxWait cost improvement (FCFS−FCFS maxes at 369403 s.)
6e+05
4e+05
2e+05
0e+00
−5000
−2500
0
2500
5000
Average AvgWait cost improvement (FCFS−FCFS averages at 6952 s.)
strongly worded e-mail
~30%
Backfilling
exp
fcfs
lcfs
lpf
lqf
spf
sqf
Primary
exp
fcfs
lcfs
lpf
lqf
spf
sqf

We recover no-starvation guarantees by using a threshold.

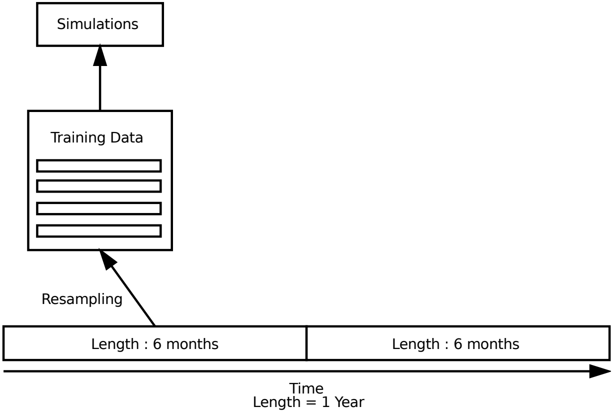
**if** $\text{wait}_j > T$ **then**
  Push job $j$ ahead of the wait queue.
**end if**

# Thresholding: Simulation results with 20h.



4 Experimental validation

- Train/test experiments.
- Methodology
- Traces
- Results


Simulations
Training Data
Resampling
Length : 6 months
Length : 6 months
Time
Length = 1 Year

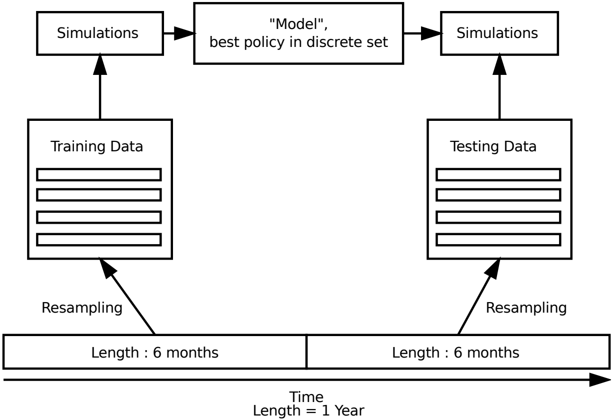
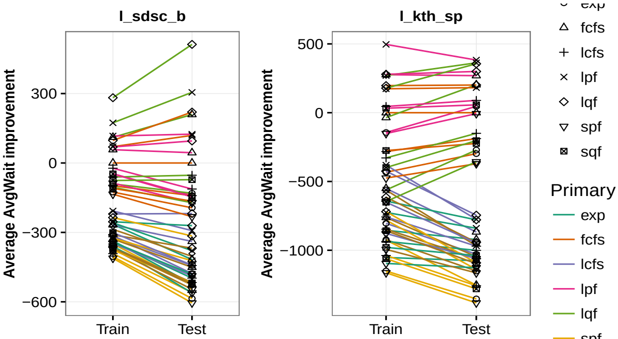

Simulations
"Model",
best policy in discrete set
Simulations
Training Data
Testing Data
Resampling
Resampling
Length : 6 months
Length : 6 months
Time
Length = 1 Year

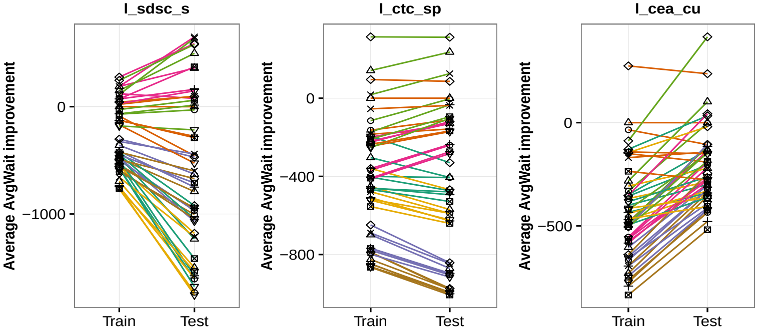
Table: Workload logs used in the simulations.

| Name | Year | Processors | Jobs | Duration |
|---|---|---|---|---|
| KTH-SP2 | 1996 | 100 | 28k | 11 Months |
| CTC-SP2 | 1996 | 338 | 77k | 11 Months |
| SDSC-SP2 | 2000 | 128 | 59k | 24 Months |
| SDSC-BLUE | 2003 | 1,152 | 243k | 32 Months |
| CEA-Curie | 2012 | 80,640 | 312k | 3 Months |




**EASY−FCFS−FCFS value:**
**Train Avg: 1768.1**
**Test Avg: 2399.0**
**Test Max:129789.000000**
**Learned Max:**
**80612.000000**
**Avg Decrease : −25.21%**

**EASY−FCFS−FCFS value:**
**Train Avg: 6952.5**
**Test Avg: 4409.0**
**Test Max:204919.000000**
**Learned Max:**
**186764.000000**
**Avg Decrease : −31.36%**




| **l_sdsc_s** | **l_ctc_sp** | **l_cea_cu** |
|---|---|---|
| **EASY−FCFS−FCFS value**<br>**Train Avg: 4242.5**<br>**Test Avg: 13592.3**<br>**Test Max:142532.000000**<br>**Learned Max: 61694.000000**<br>**Avg Decrease : −12.73%** | **EASY−FCFS−FCFS value:**<br>**Train Avg: 2349.8**<br>**Test Avg: 2676.2**<br>**Test Max:96178.000000**<br>**Learned Max: 92365.000000**<br>**Avg Decrease : −37.62%** | **EASY−FCFS−FCFS value:**<br>**Train Avg: 1707.7**<br>**Test Avg: 1106.1**<br>**Test Max:141804.000000**<br>**Learned Max: 102942.000000**<br>**Avg Decrease : −46.89%** |

## Conclusion

Adaptive policies are possible in batch scheduling!

We can reduce the waiting time from 12 to 46 percent on average.

**Conclusion**

Adaptive policies are possible in batch scheduling!
We can reduce the waiting time from 12 to 46 percent on average.

- This requires simulation. Can we eliminate this requirement?
  - Multi-armed bandit.
- Can we be more ambitious?
  - Wider search space
  - Contextual policy choice

Gaussier, É., Glesser, D., Reis, V., and Trystram, D. (2015). Improving backfilling by using machine learning to predict running times. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015*, pages 64:1–64:10.

Gaussier, É., Lelong, J., Reis, V., and Trystram, D. (2017). Online (bandit) policy selection for easy-backfilling. In *IN SUBMISSION, Supercomputing 2017.*

Lelong, J., Reis, V., and Trystram, D. (2017). Tuning backfilling queues. In *Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS 2017, Orlando, Florida, USA*, pages 64:1–64:10.

Ngoko, Y., Trystram, D., Reis, V., and Cérin, C. (2016). An automatic tuning system for solving np-hard problems in clouds. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2016, Chicago, IL, USA, May 23-27, 2016*, pages 1443–1452.

This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) funded by the

French program Investissements d'avenir.