Efforts to Bridge Theory and Practice on Distributed Scheduling Algorithms

Initial research results

Laércio Lima Pilla, Johanne Cohen

pilla@lri.fr

Agenda

Motivation

Approach

Evaluation

Conclusion

Motivation

Increase in scale of HPC application and platforms → more potential for issues with load imbalance *(dynamic behavior, performance variations, failures …)*

© Cyril FRESILLON / IDRIS / CNRS Photothèque

Motivation

Increase in scale of HPC application and platforms ⟶ more potential for issues with load imbalance *(dynamic behavior, performance variations, failures ...)*

Solution for load imbalance at run-time: dynamic load balancing

Problem with usual dynamic load balancing: overhead of centralizing information, sequential decisions

Motivation

How to avoid centralizing information? Distributed load balancing.

Examples:

- RTSs with work stealing (e.g., PaRSEC)
- Lifflander et al. *Work Stealing and Persistence-based Load Balancers for Iterative Overdecomposed Applications* (HPDC 2012)
- Menon, Kale. *A Distributed Dynamic Load Balancer for Iterative Applications* (SC 2013)
- Freitas et al. *A Batch Task Migration Approach for Decentralized Global Rescheduling* (SBAC-PAD 2018)
- Pebay, Lifflander. *Distributed Load Balancing Utilizing the Communication Graph* (Charm++ Workshop 2019)
- Freitas et al. *PackStealLB: A Scalable Distributed Load Balancer based on Work Stealing and Workload Discretization* (Pre-print)

Motivation

How to avoid centralizing information? Distributed load balancing.

Examples:

- RTSs with work stealing (e.g., PaRSEC)
- Lifflander et al. *Work Stealing and Persistence-based Load Balancers for Iterative Overdecomposed Applications* (HPDC 2012)
- Menon, Kale. *A Distributed Dynamic Load Balancer for Iterative Applications* (SC 2013)
- Freitas et al. *A Batch Task Migration Approach for Decentralized Global Rescheduling* (SBAC-PAD 2018)
- Pebay, Lifflander. *Distributed Load Balancing Utilizing the Communication Graph* (Charm++ Workshop 2019)
- Freitas et al. *PackStealLB: A Scalable Distributed Load Balancer based on Work Stealing and Workload Discretization* (Pre-print)

… not many distributed algorithms used in practice

Motivation

What about searching in the field of distributed algorithms?

- Balls into bins problems
- Game theory (e.g., no-regret learning)
- Average consensus
- …

Motivation

What about searching in the field of distributed algorithms?

- Balls into bins problems
- Game theory (e.g., no-regret learning)
- Average consensus
- ...

Abstract models, little to no information

Search for bounds, convergence, Nash equilibria ...

Motivation

Objective: bridge the gap between theory and practice on distributed scheduling algorithms

Approach: search for ways to adapt and improve distributed scheduling algorithms from the literature to be used in HPC scenarios

Approach

Take an algorithm from the literature and expand on it.

Berenbrink et al. *Distributed Selfish Load Balancing* (SIAM Journal on Computing, 2007).

Structure of a round:

For each task b do in parallel:

Let i_b be the current resource of task b

Choose resource j_b uniformly at random

Let $X_{ib}(t)$ be the current load of resource i

Let $X_{jb}(t)$ be the current load of resource j

If $X_{ib}(t) > X_{jb}(t)$ then:

Move task b from i_b to j_b with probability $1 - X_{jb}(t)/X_{ib}(t)$

- ***m* atomic, unitary tasks**
- ***n* identical resources**
- **ε-Nash equilibrium in O(loglog *m*)**
- **Perfect balance expected in O(loglog m+n^4)**

Approach

Information to add: average resource load ($\bar{x} = m/n$)

Organize the resources in three categories:

- *Underloaded*: $X_{\{i,j\}}(t) < \bar{x}$
- *Average-loaded*: $\bar{x} <= X_{\{i,j\}}(t) <= \bar{x} + \varepsilon$
- *Overloaded*: $\bar{x} + \varepsilon < X_{\{i,j\}}(t)$

Use this information to define which resources can send tasks (*sources*) and receive tasks (*destinations*)

Approach

Variations over Selfish + average load

Source \ Dest	Inf (only underloaded)	Avg (average-load and below)	Sup (all resources)
Sup (only overloaded)	Sup_Inf (most restrictive)	Sup_Avg	Sup_Sup
Avg (average-load or above)	Avg_Inf	Avg_Avg	Avg_Sup
Inf (all resources)	Inf_Inf	Inf_Avg	Inf_Sup (original)

Approach

Variations over Selfish + average load

Structure of a round for SelfishAL_Sup_Inf:

For each task b in an overloaded resource do in parallel:

Let i_b be the current resource of task b

Choose resource j_b uniformly at random

Let $X_{ib}(t)$ be the current load of resource i

Let $X_{jb}(t)$ be the current load of resource j

If j_b is underloaded ($X_{jb}(t) < \bar{x}$) then:

Move task b from i_b to j_b with probability $1 - X_{jb}(t)/X_{ib}(t)$

Source \ Dest	Inf (only underloaded)	Avg (average-load and below)	Sup (all resources)
Sup (only overloaded)	Sup_Inf (most restrictive)	Sup_Avg	Sup_Sup
Avg (average-load or above)	Avg_Inf	Avg_Avg	Avg_Sup
Inf (all resources)	Inf_Inf	Inf_Avg	Inf_Sup (original)

Evaluation

Based on a simple simulator @ https://github.com/llpilla

Why simulation?

- The number of resources needed to obtain results can be large
- Easier debugging
- Easier to reproduce results

Metrics

- Number of rounds for convergence
- Number of task migrations
- Number of load checks

Evaluation

Hypotheses

1. When compared to the original Selfish algorithm, the addition of the average resource load information decreases the number of rounds necessary for convergence.
2. Limiting the set of source resources decreases the total number of task migrations.
3. Limiting the set of source resources decreases the total number of load checks.
4. Limiting the set of destination resources decreases the total number of task migrations.
5. Limiting the set of destination resources decreases the total number of load checks.
6. The original task distribution in a scheduling scenario affects the number of rounds an algorithm takes to converge.
7. The higher the number of resources in a scheduling scenario, the higher the number of rounds it takes to balance the load.

Evaluation

Additional details about the simulation

- All tasks make decisions based on the information available at the start of the round
  - *No worries about ordering messages*
- The simulation is stopped if an algorithm does not converge in 1000 rounds
- Convergence = no resource has a load 5% over the average resource load

Evaluation

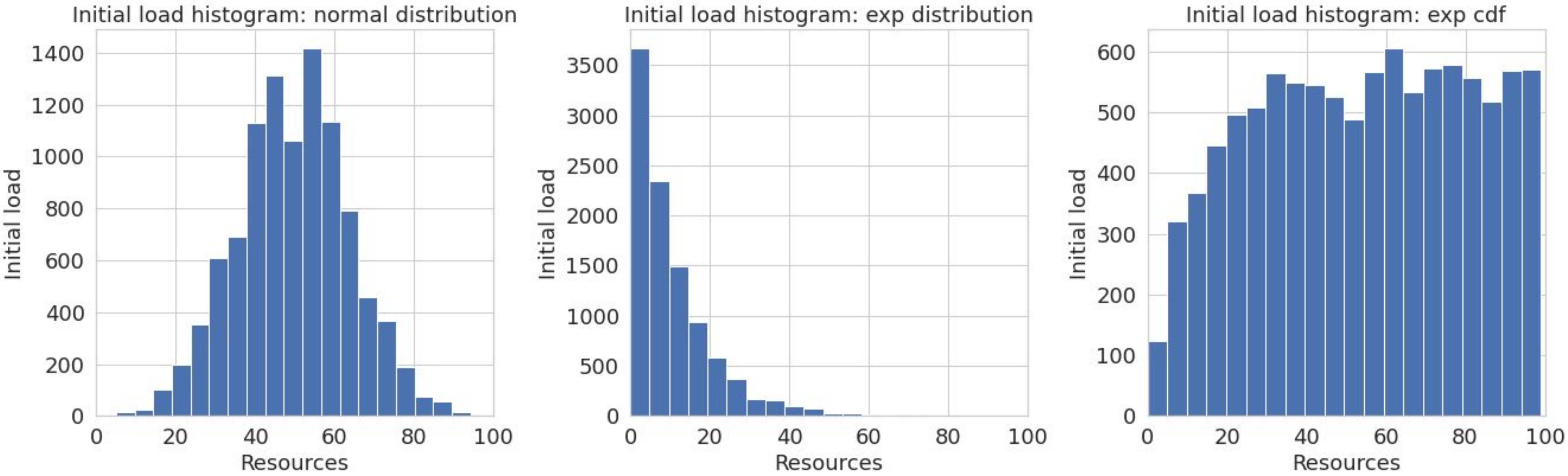
Simulation parameters

Parameter	Values	Total number
Number of resources	10, 50, 100, 500	4
Average number of tasks per resource	50, 100	2
Type of tasks	Unitary	1
Initial task distribution	1 with all tasks in resource 0	31
	10 normal	
	10 exponential	
	10 exponential CDF	

248 scenarios x 9 schedulers x 10 samples = 22320 samples

Evaluation

Examples of initial load distributions (100 resources, 10000 tasks)



248 scenarios x 9 schedulers x 10 samples = 22320 samples

Evaluation

Hyp. 1: When compared to the original Selfish algorithm, the addition of the average resource load information decreases the number of rounds necessary for convergence.

Mean number of rounds for convergence

Source \ Dest	Inf (only underloaded)	Avg (average-load and below)	Sup (all resources)
Sup (only overloaded)	12.956	4.281	4.732
Avg (average-load or above)	21.606	4.191	4.541
Inf (all resources)	11.896	33.761	41.677

Evaluation

Hyp. 1: When compared to the original Selfish algorithm, the addition of the average resource load information decreases the number of rounds necessary for convergence.

Mean number of rounds for convergence

Source \ Dest	Inf (only underloaded)	Avg (average-load and below)	Sup (all resources)
Sup (only overloaded)	12.956	4.281	4.732
Avg (average-load or above)	21.606	4.191	4.541
Inf (all resources)	11.896	33.761	41.677

Number of cases that did not converge in 1000 rounds

Inf_Sup: 9 times
Inf_Avg: 7 times

Evaluation

Hyp. 1: When compared to the original Selfish algorithm, the addition of the average resource load information decreases the number of rounds necessary for convergence.

Mean number of rounds when converged

Source \ Dest	Inf (only underloaded)	Avg (average-load and below)	Sup (all resources)
Sup (only overloaded)	12.956	4.281	4.732
Avg (average-load or above)	21.606	4.191	4.541
Inf (all resources)	11.896	33.761 31.023	41.677 38.182

Evaluation

Hyp. 1: When compared to the original Selfish algorithm, the addition of the average resource load information decreases the number of rounds necessary for convergence.

Mean number of rounds when converged

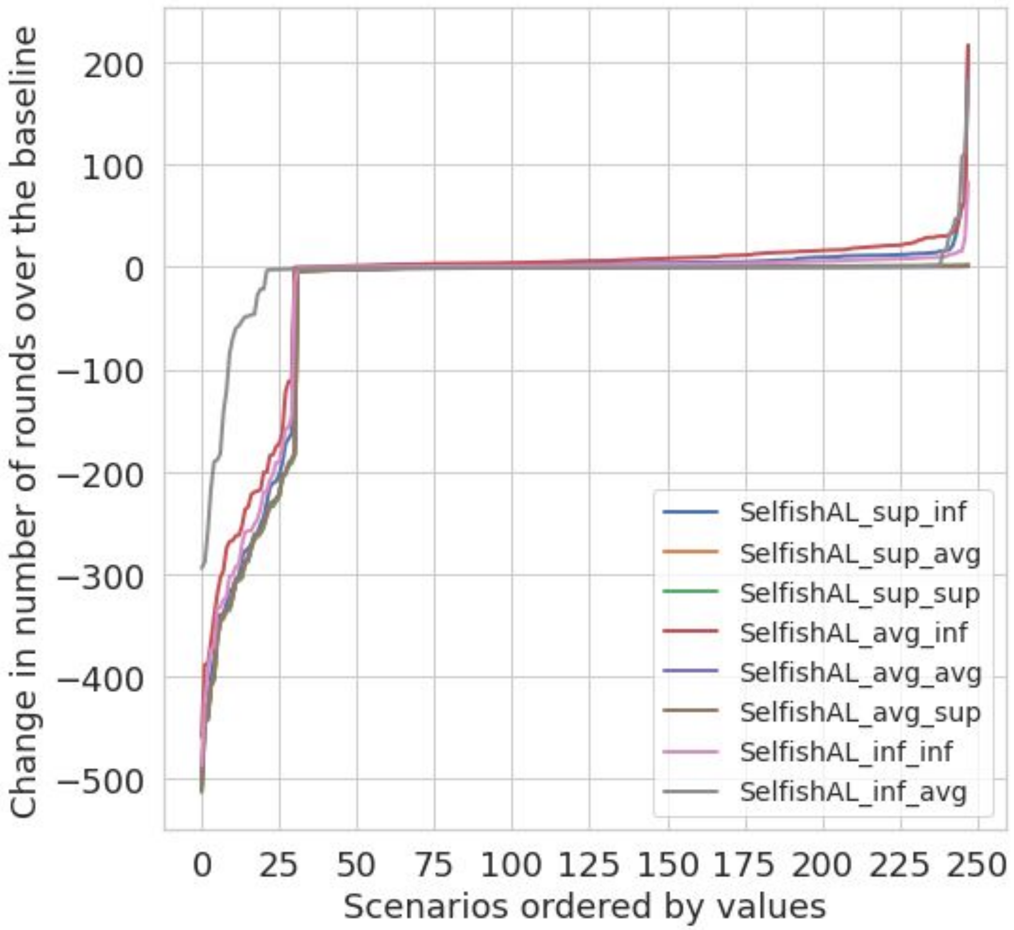
Source \ Dest	Inf (only underloaded)	Avg (average-load and below)	Sup (all resources)
Sup (only overloaded)	12.956	4.281	4.732
Avg (average-load or above)	21.606	4.191	4.541
Inf (all resources)	11.896	31.023	38.182

Median number of rounds for convergence

Source \ Dest	Inf (only underloaded)	Avg (average-load and below)	Sup (all resources)
Sup (only overloaded)	9	4	5
Avg (average-load or above)	13	3	4
Inf (all resources)	7	3	4

Evaluation

Hyp. 1: When compared to the original Selfish algorithm, the addition of the average resource load information decreases the number of rounds necessary for convergence.

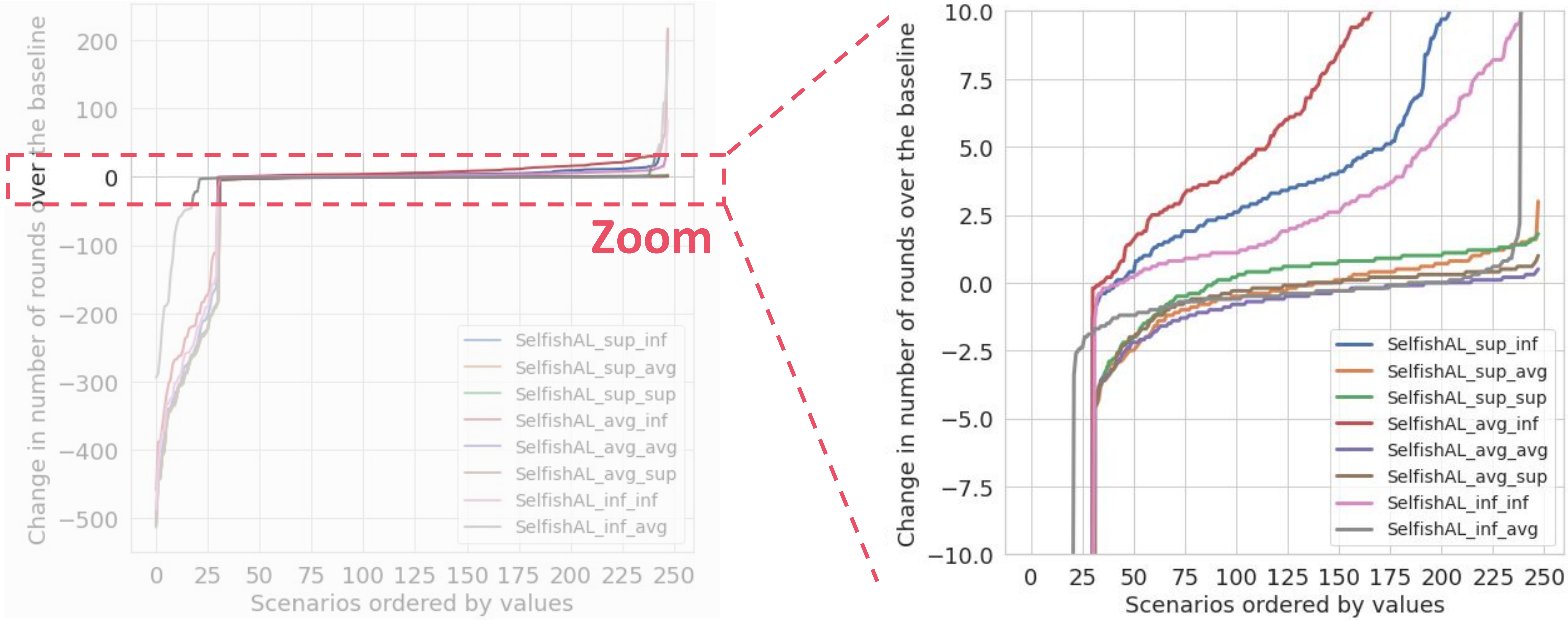


Comparison between variations and the original Selfish

Value of a scenario =
avg. rounds for variation
- avg. rounds for Selfish

Evaluation

Hyp. 1: When compared to the original Selfish algorithm, the addition of the average resource load information decreases the number of rounds necessary for convergence.



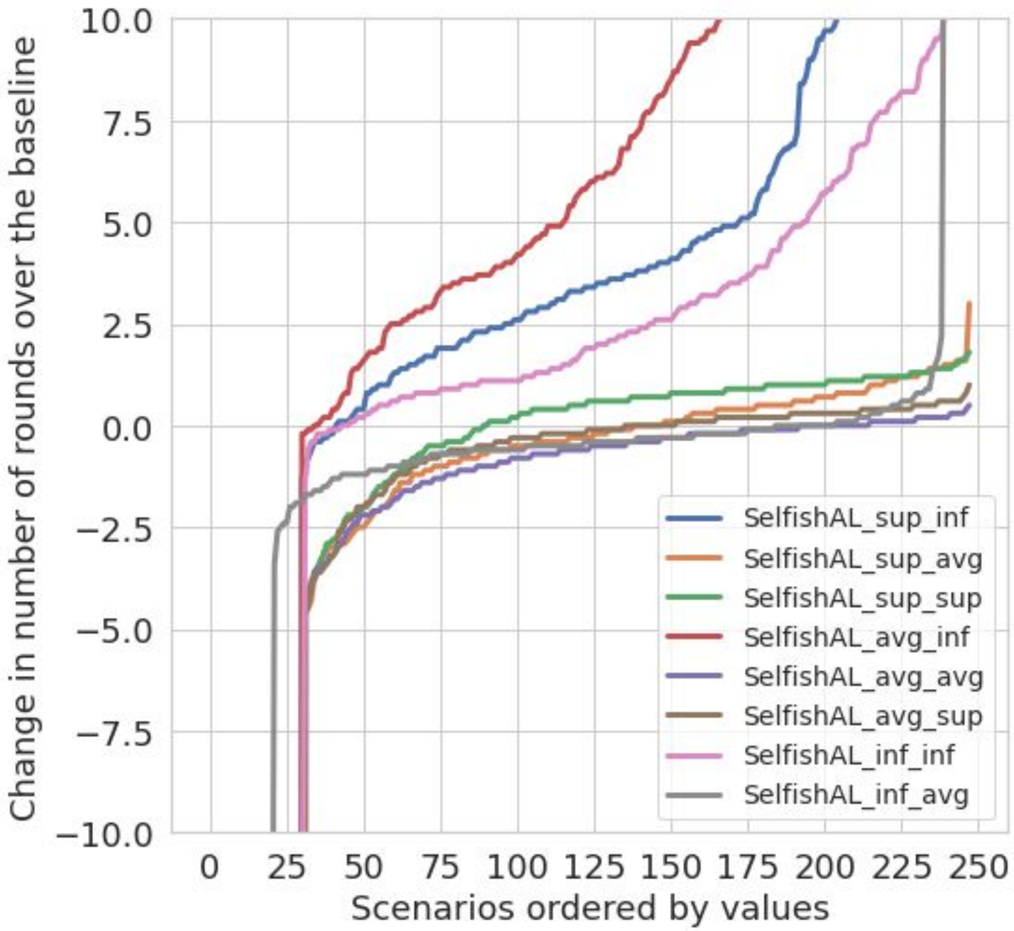
Evaluation

Hyp. 1: When compared to the original Selfish algorithm, the addition of the average resource load information decreases the number of rounds necessary for convergence.

Extra lessons

Using only underloaded destinations -> bad idea

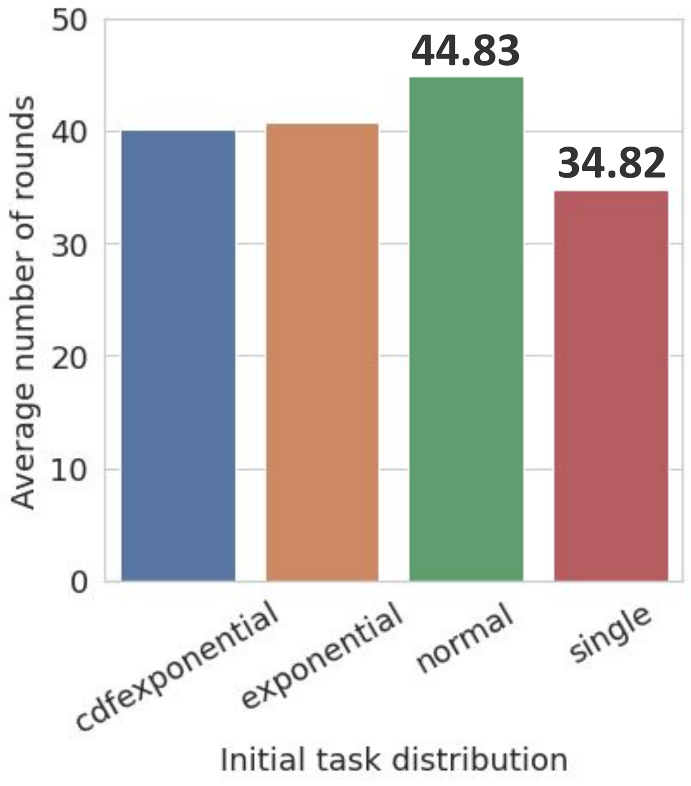
Using overloaded and average-loaded as sources -> it *can be* a good idea



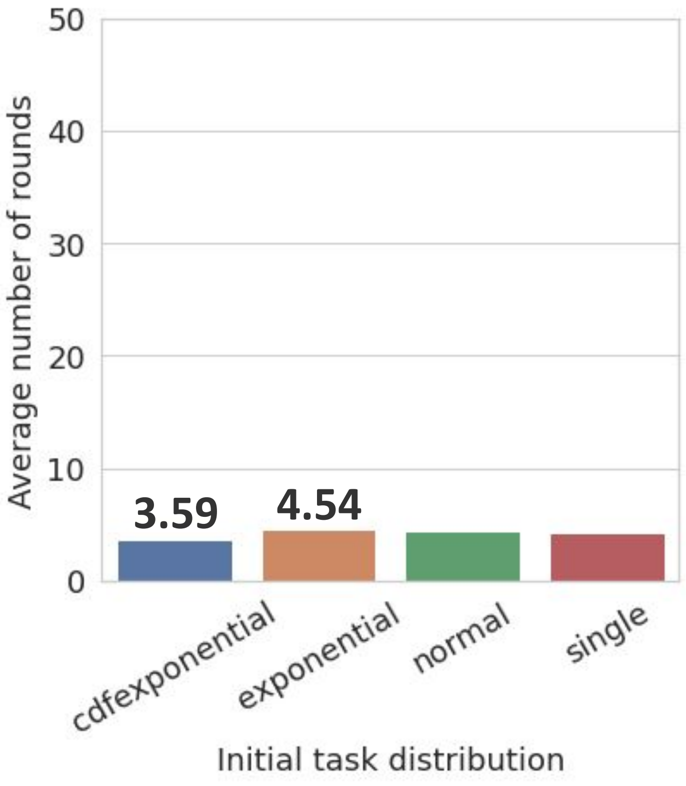
Evaluation

Hyp. 6: The original task distribution in a scheduling scenario affects the number of rounds an algorithm takes to converge.

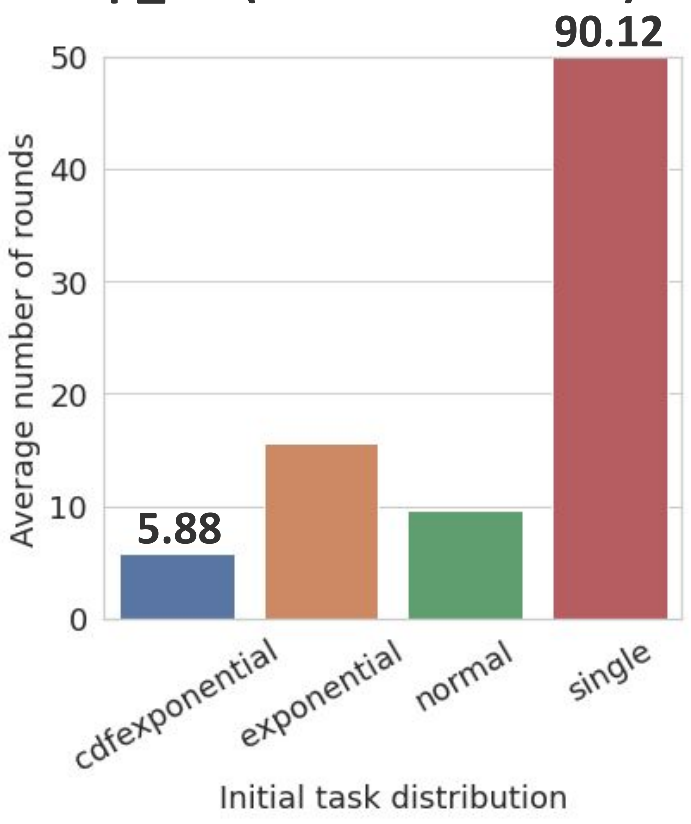
Inf_Sup (original)



Avg_Avg



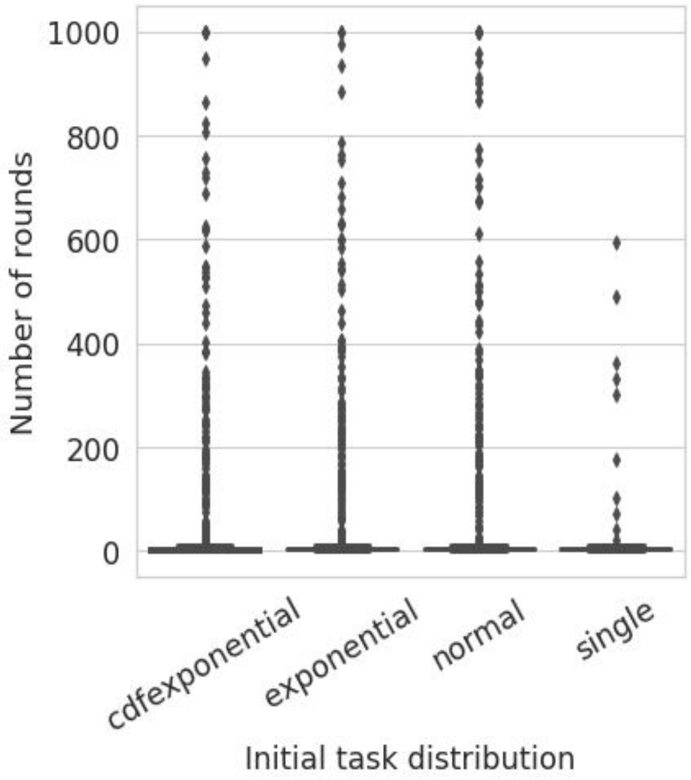
Sup_Inf (most restrictive)



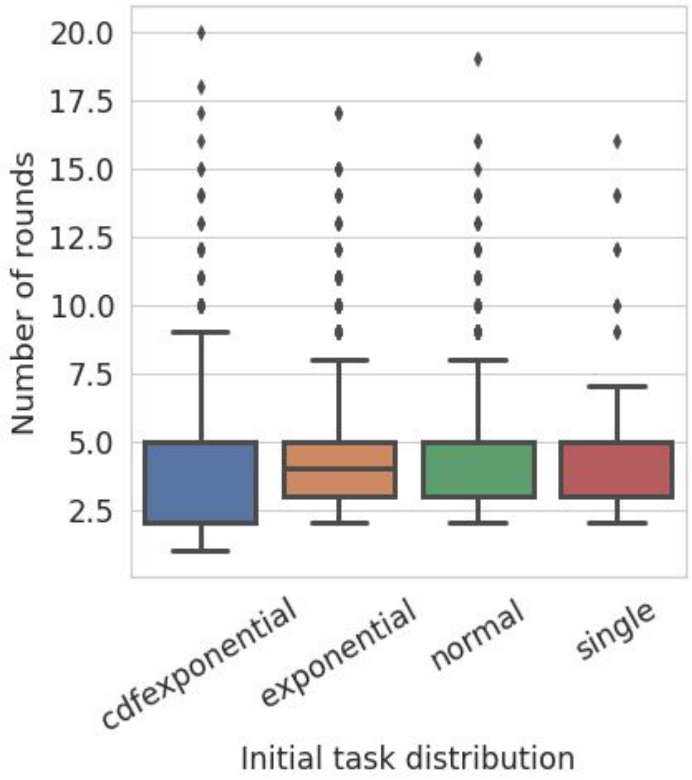
Evaluation

Hyp. **6: The original task distribution in a scheduling scenario affects the number of rounds an algorithm takes to converge.**

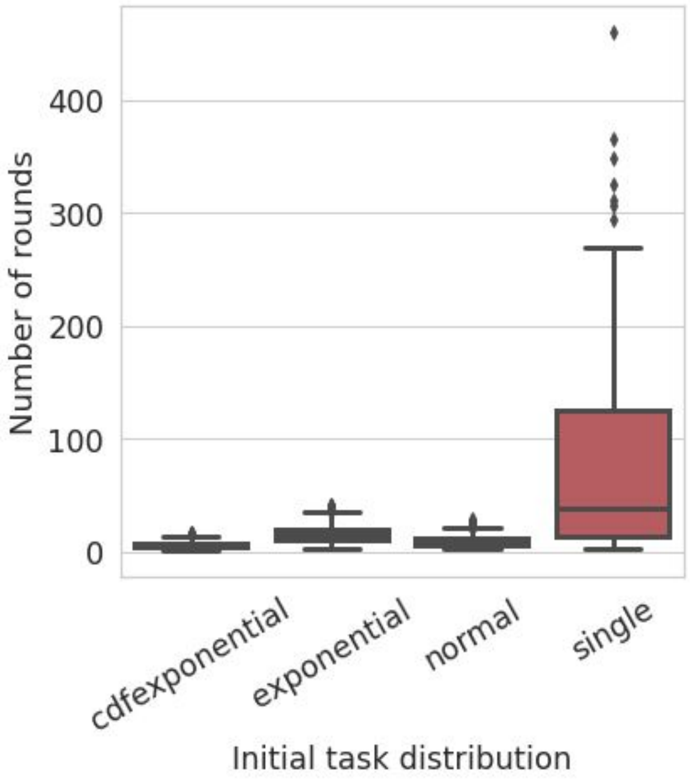
Inf_Sup (original)



Avg_Avg



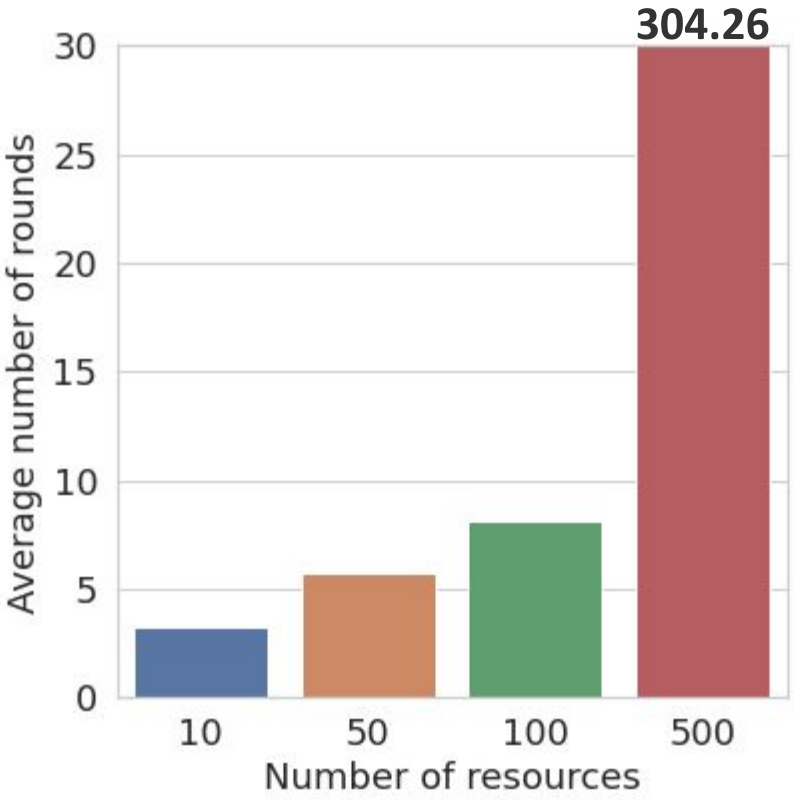
Sup_Inf (most restrictive)



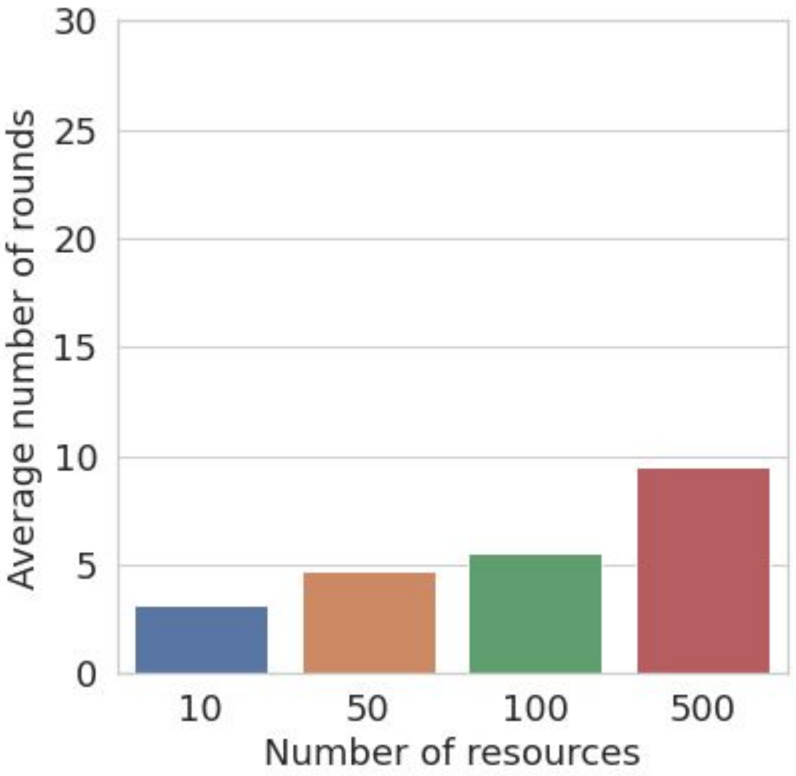
Evaluation

Hyp. 7: **The higher the number of resources in a scheduling scenario, the higher the number of rounds it takes to balance the load.**

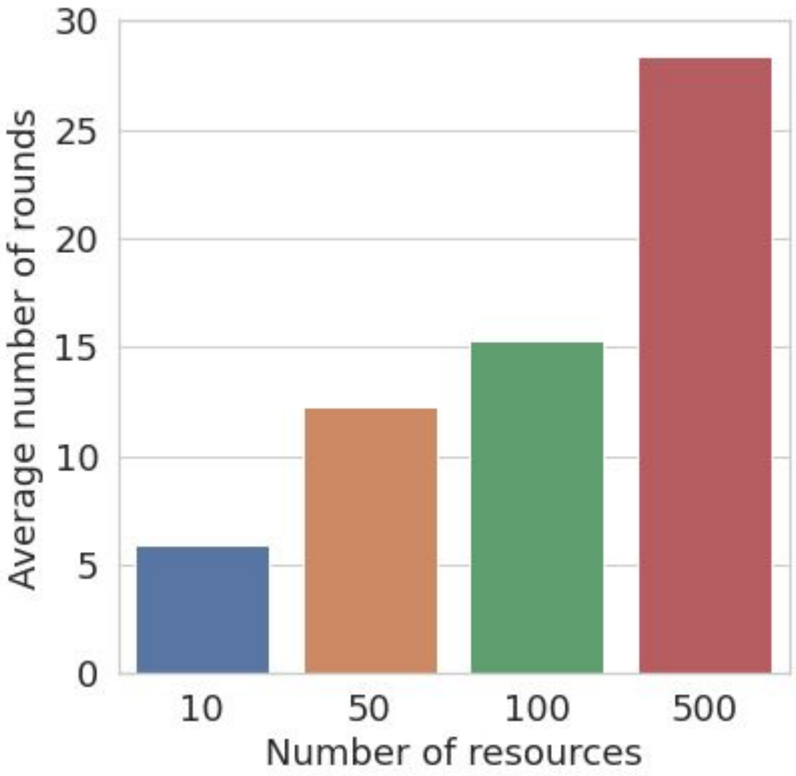
Inf_Sup (original)



Avg_Avg - 50 tasks per resource



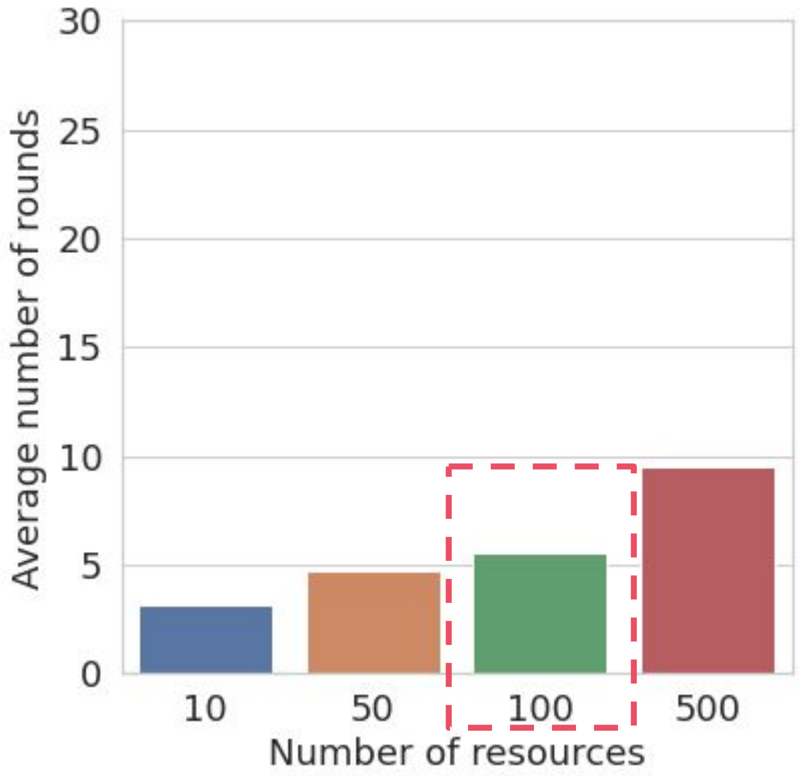
Sup_Inf (most restrictive)



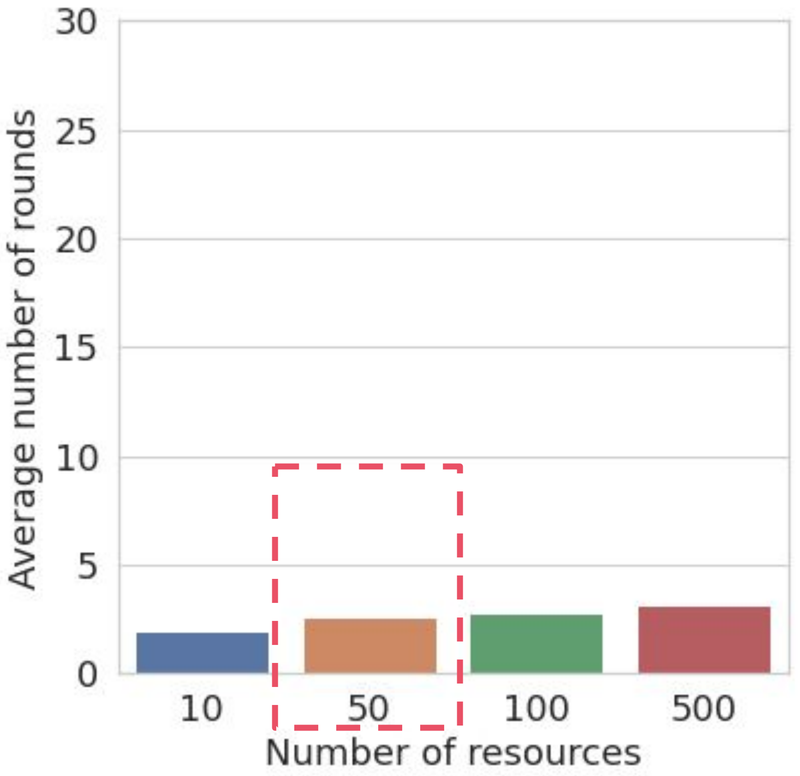
Evaluation

Hyp. 7: The **higher the number of resources** in a scheduling scenario, the higher the number of rounds it takes to balance the load.

Avg_Avg - 50 tasks per resource



Avg_Avg - 100 tasks per resource



5000 tasks

Conclusion

First steps to bridge the gap between theory and practice.

Hypotheses hold so far.

Next steps

- Run experiments in larger scale
- Run experiments with non-unitary tasks
- Add more information and techniques
- Test in real HPC systems after filtering variations

Conclusion

Final remarks

We need standard load balancing simulators. *Do you know of any?*

- Cannot always count on a supercomputer reservation
- Make it easier to compare algorithms and results

We need standard load balancing benchmarks with well-documented parameters.

- Task Bench is a nice step in the right direction but it is not focused in load balancing

Efforts to Bridge Theory and Practice on Distributed Scheduling Algorithms

Initial research results

Thank you.

Laércio Lima Pilla, Johanne Cohen

pilla@lri.fr