IO-SETS: Simple and efficient approaches for I/O bandwidth management

Francieli Boito, Guillaume Pallez, **Luan Teylo**, Nicolas Vidal

Data-Aware Scheduling at Higher Scale (DASH)

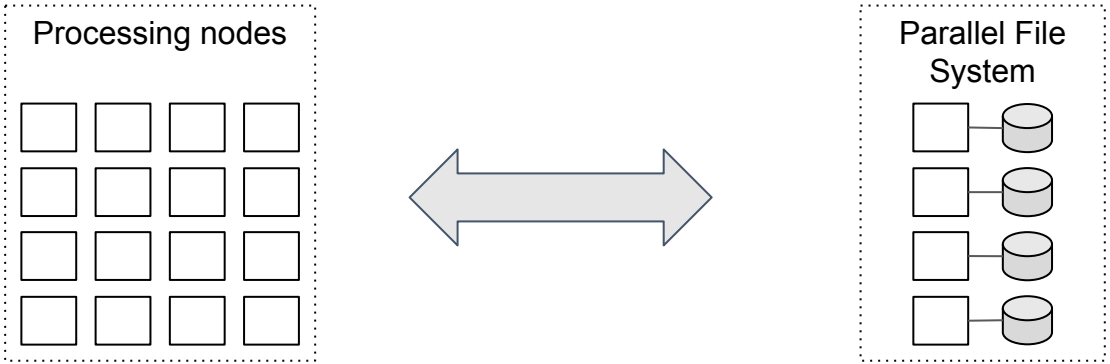
- **Three papers and one research report:**
  - The role of storage target allocation in BeeGFS (CLUSTER 2022)
    - https://gitlab.inria.fr/hpc_io/beegfs_evaluation
  - I/O performance of multiscale finite element simulations (Workshop in SBAC-PAD 2022)
    - https://gitlab.inria.fr/hpc_io/iofwd_perf_impact
  - IO-SETS (Submitted to TPDS)
    - https://gitlab.inria.fr/hpc_io/io-sets
  - Implementation of a Weighted Fair Queuing (research report)
    - https://github.com/francielizanon/agios

- **Highlights:**
  - We recommended a different configuration for the PFS in PlaFRIM (40% improvement in I/O performance)
  - The tools that we started to develop in this ANR/region project were then propagated to the ADMIRE H2020 European project

IO-SETS: Simple and efficient approaches for I/O bandwidth management

Context

- The I/O infrastructure is **shared by all jobs** in a supercomputer
  - “Fair-share scheduling”: applications share the bandwidth
- Performance variability due to **interference** from other applications
- Longer execution time, waste of compute resources

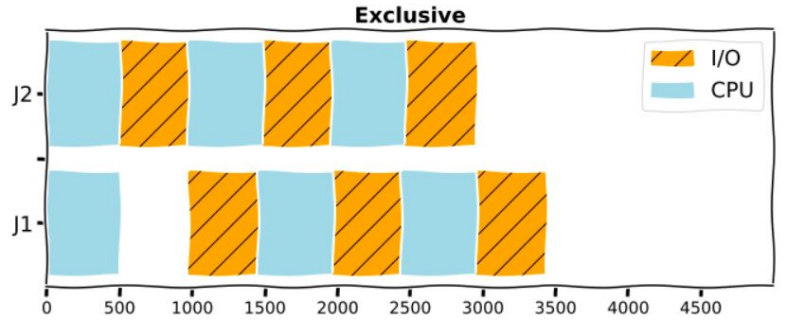


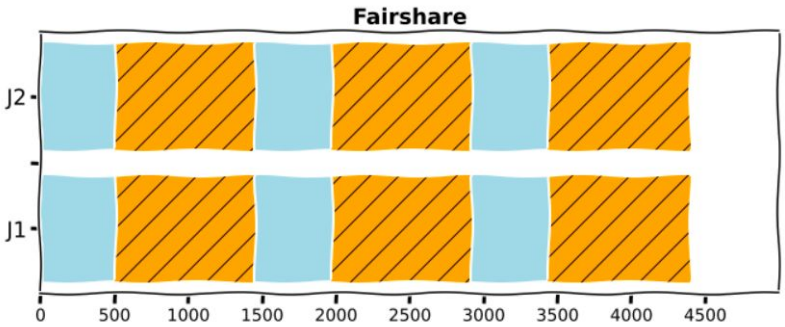
Motivation

- **I/O scheduling** to mitigate interference
  - control all accesses to the parallel file system
  - decide what applications can do I/O and when
- Most related work: **exclusive access** to the I/O infrastructure
  - requires information about application: I/O phases, amount of data, etc
- Our goal: simple scheduling heuristic
  - low cost (in computation)
  - very little information about applications

Exclusive vs. Fairshare: an example

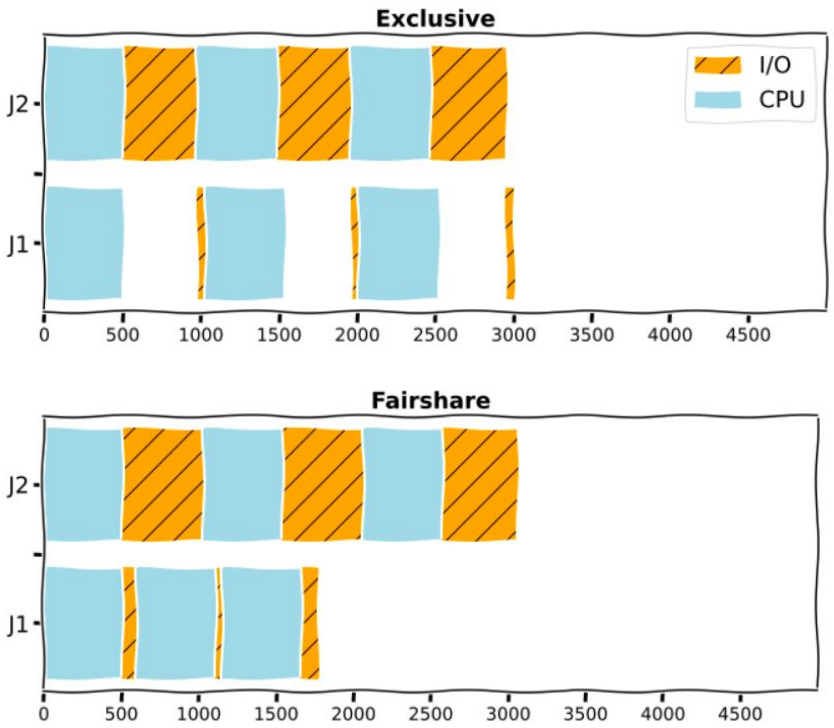
- Two concurrent periodic applications
  - “small” or “large” I/O phases





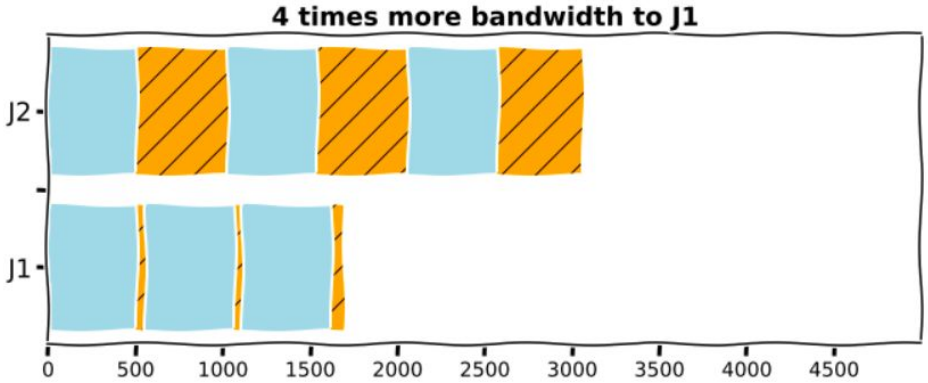
Exclusive vs. Fairshare: an example

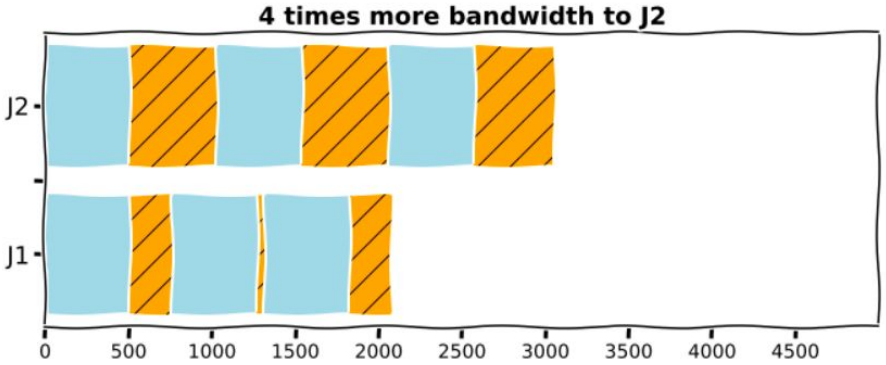
- Two concurrent periodic applications
  - “small” or “large” I/O phases



Priority-based Bandwidth sharing

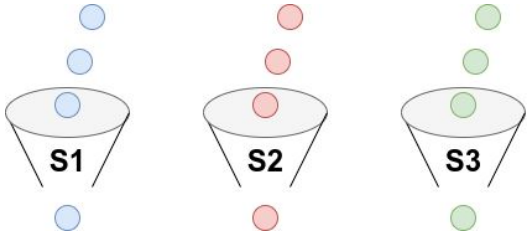
- Two concurrent periodic applications
  - “small” (**J1**) or “large” (**J2**) I/O phases





IO-Sets

- We propose IO-Sets, a set-based method
  - at the start of an I/O phase, the application is assigned to a set S_i
  - each set S_i is assigned a priority p_i
  - only one application per set is allowed to do I/O
    - exclusive access within each set
    - sharing between sets
  - the available bandwidth is shared among sets according to their priorities

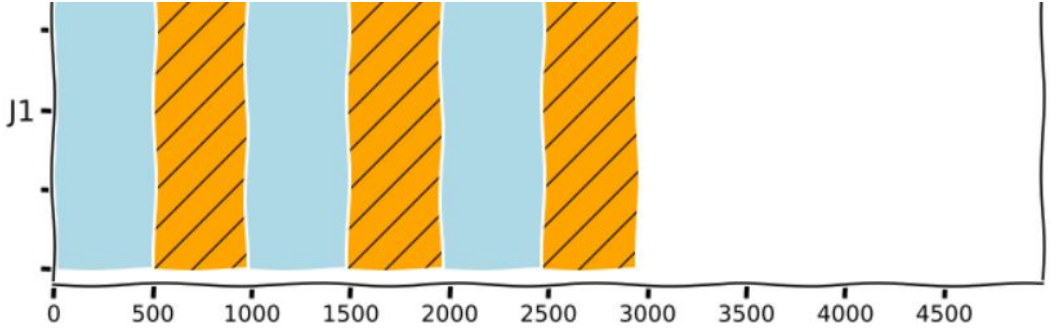


- We can propose heuristics in the IO-Sets method: answer two questions
  - **How to assign applications to sets?**
  - **How to define the priority of each set?**

Set-10 heuristic

- We define the **w_{iter}** metric for an application with n iterations
  - the average time between the beginning of two consecutive I/O phases

$$w_{\text{iter}} \stackrel{\text{def}}{=} \frac{\sum_{i \leq n} t_{\text{cpu}}^{(i)} + t_{\text{io}}^{(i)}}{n}.$$



Set-10 heuristic

- <u>Set-10 algorithm</u> in the IO-Sets method:
  - An application is assigned to a set Si that corresponds to its w_{iter} magnitude order:

$$i = \lfloor \log_{10} w_{\text{iter}}^{\text{id}} \rceil$$

  - Priorities per set decrease exponentially. Set S_i has priority p_i:

$$p_i = 10^{-i}$$

- Applications with the **smallest w_{iter} get the highest priority**, i.e. most of the bandwidth
  - { (S1, 0.1), (S2, 0.01), (S3, 0.001)….}

$$x_i = \frac{p_i}{\sum_{j=1}^{k} p_j}.$$

$$x_1 = \frac{0.1}{(0.1 + 0.01 + 0.001)} = 0.90$$

$$x_2 = \frac{0.01}{(0.1 + 0.01 + 0.001)} = 0.09$$

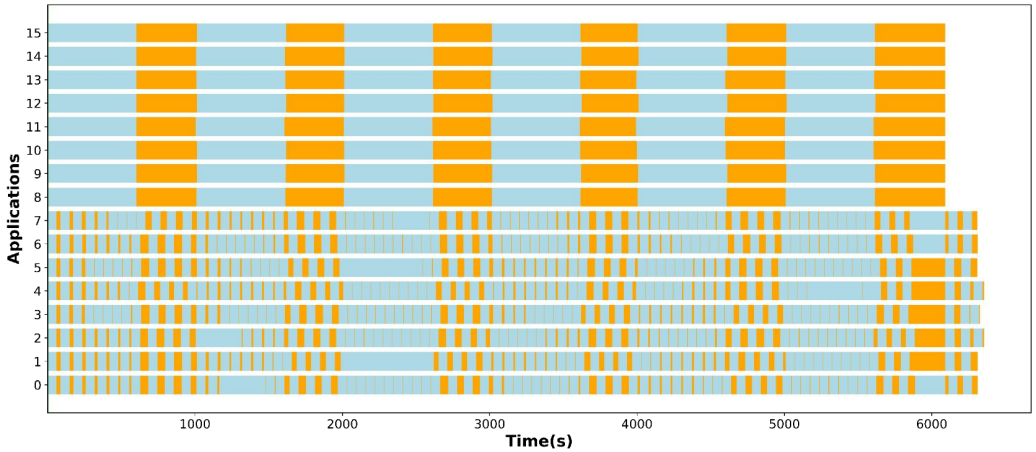
$$x_3 = \frac{0.001}{(0.1 + 0.01 + 0.001)} = 0.009$$

Evaluation

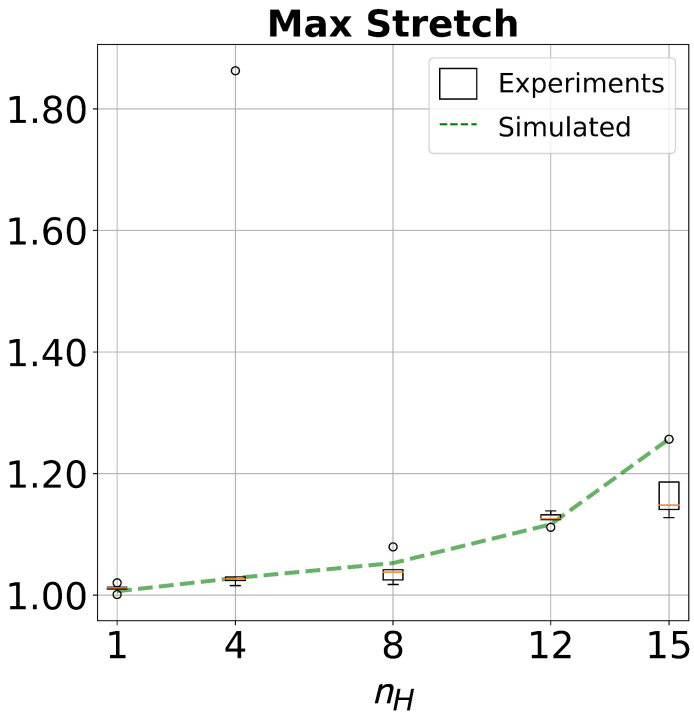
- Max Stretch: how many times slower does the slowest application run (compared to running by itself)
  - The lower the better
- Utilization: It is a system-wide metric that represents platform usage
  - The highest the better
- IO-slowdown: how close from the minimum the actual I/O time is
  - Equal 1 mean the I/O was the same as it would be in isolation

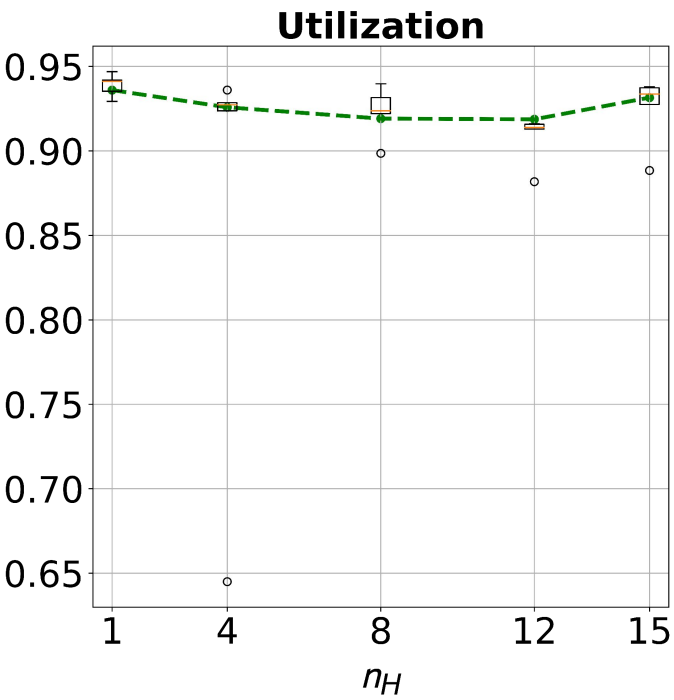
Evaluation: practical

- Practical experiments with IOR
- Simulated experiments with SimGrid
- 16 applications
  - nH high-frequency jobs with witer = 64
  - nL low-frequency jobs with witer = 640



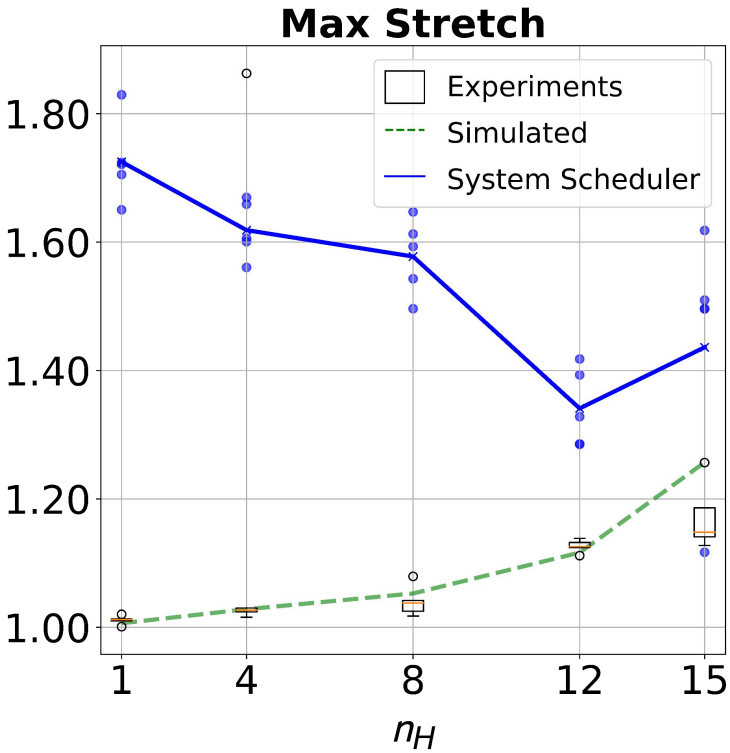
Validation

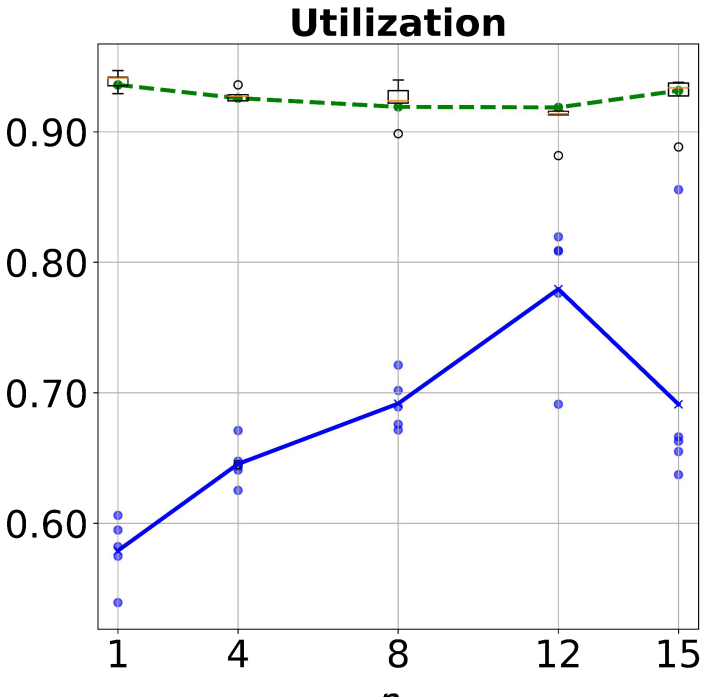




80% of the experimental results are within 3.5% (resp. 1.5%) of the simulated Utilization (resp. Max Stretch).

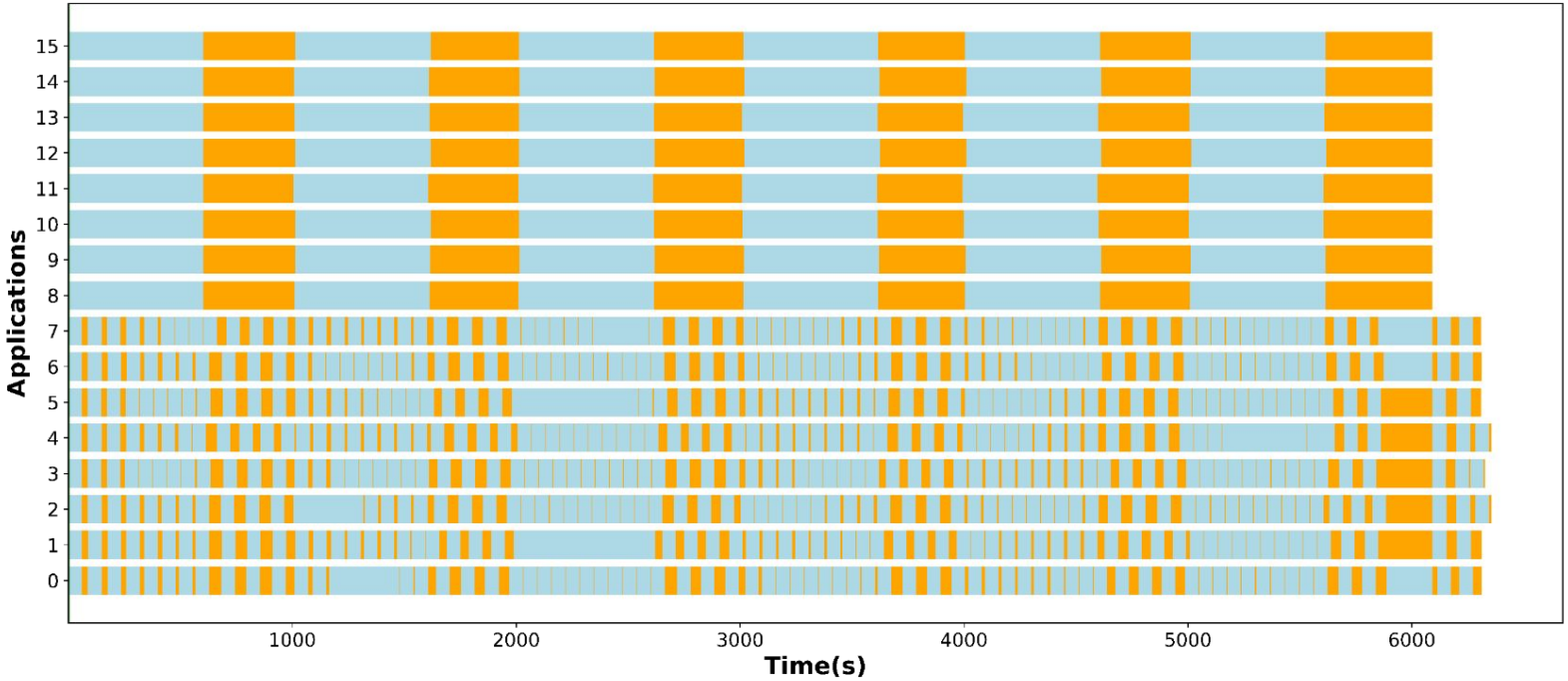
Validation



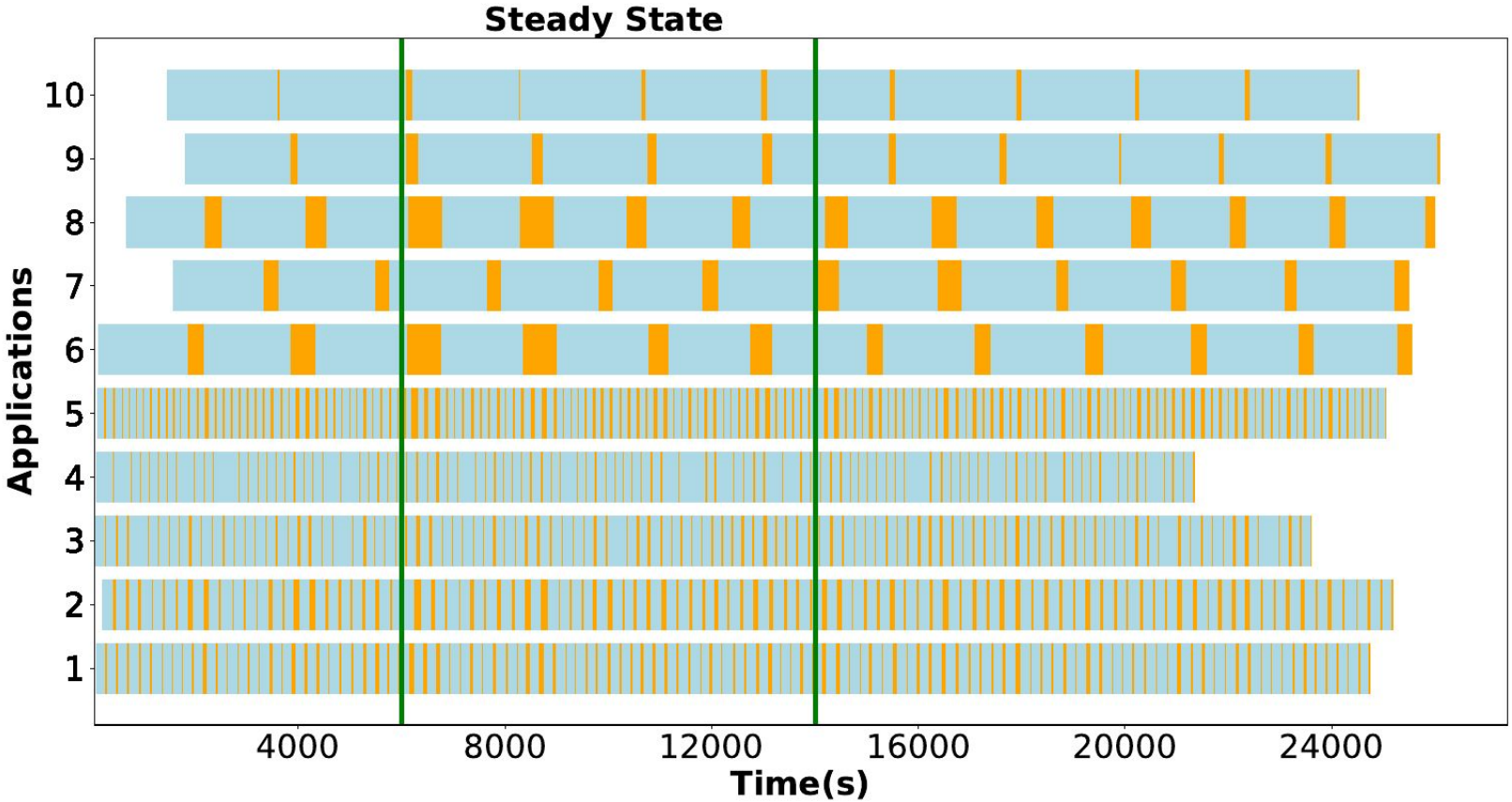


Set-10 improved the stretch by more than 27% over the system current scheduler

Validation

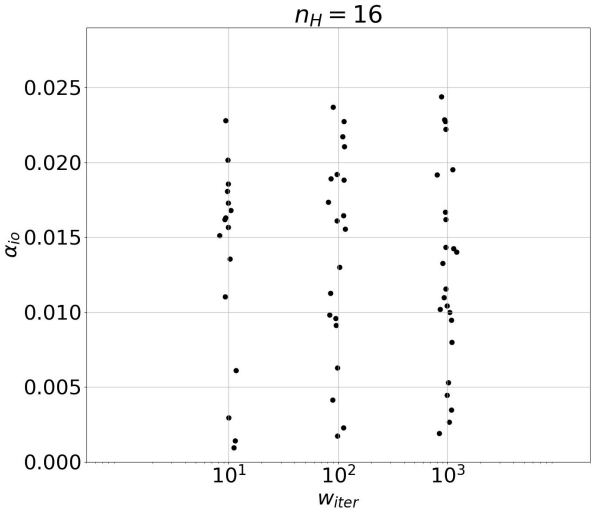

Applications
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
1000
2000
3000
4000
5000
6000
Time(s)

Validation


Steady State
Applications
10
9
8
7
6
5
4
3
2
1
4000
8000
12000
16000
20000
24000
Time(s)

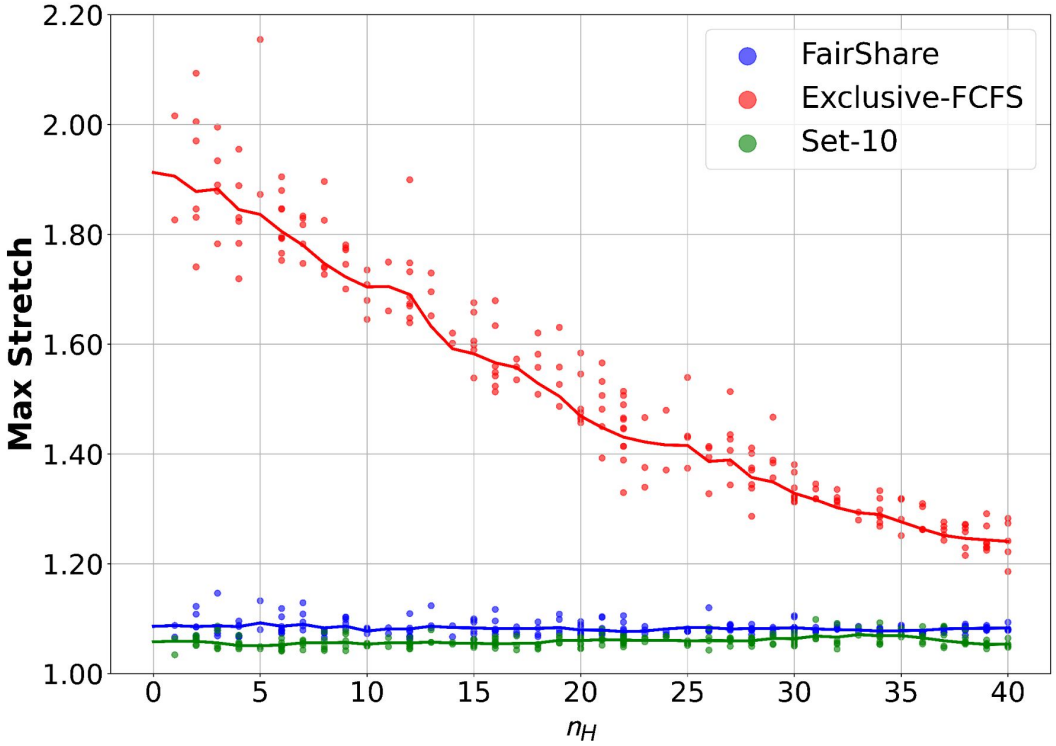
Evaluation

- Simulated experiments with SimGrid
- >200 workloads, each of 60 applications
  - nH high-frequency jobs with witer ~ N(10,1)
  - nM medium-frequency jobs with witer ~ N(100,10)
  - nL low-frequency jobs with witer ~ N(1000, 100)
- For each application, we define a random release time
- For nH = {0,...,40}, nM=20 and nL = 40 - nH



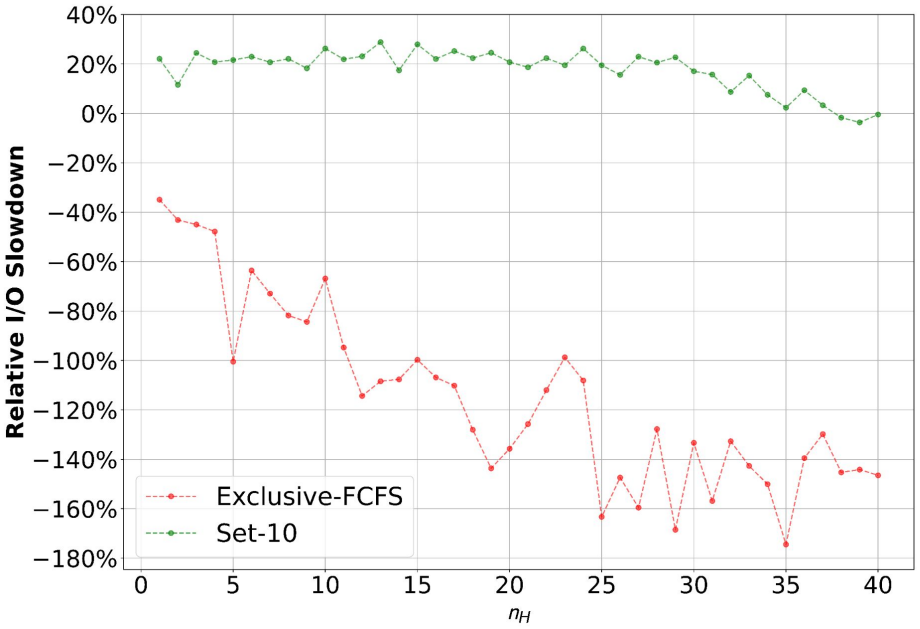
Validation

many
low-frequency
(long phases)
applications



many
high-frequency
(short phases)
applications

I/O performance impact



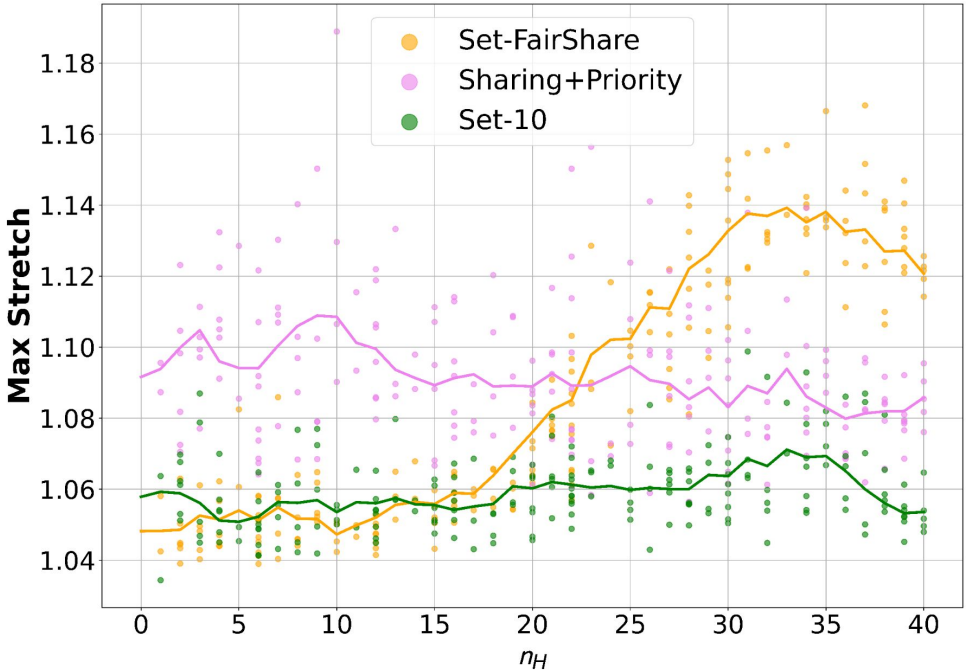
I/O slowdown of Set-10 over FairShare: improvement of up to 25%

many low-frequency (long phases) applications

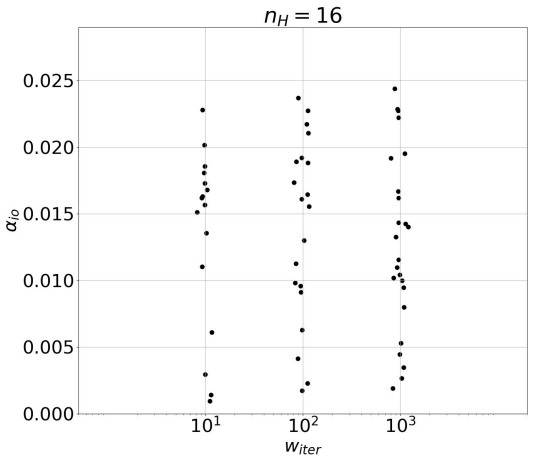
many high-frequency (short phases) applications

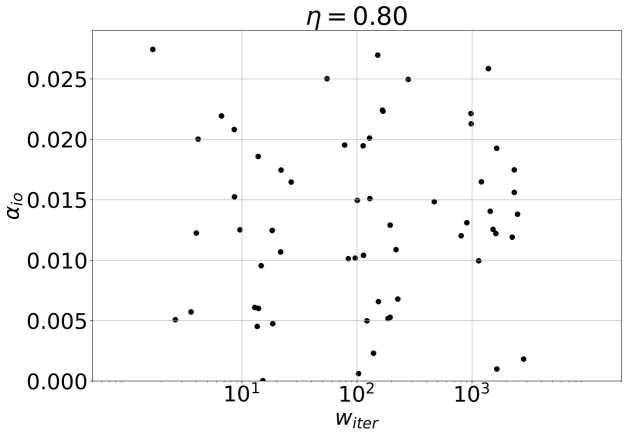
Where do results come from?

Compared to only having sets (“Set-Fairshare”) and priority-based bandwidth without sets (“Sharing+Priority”)



If the sets are not well-defined

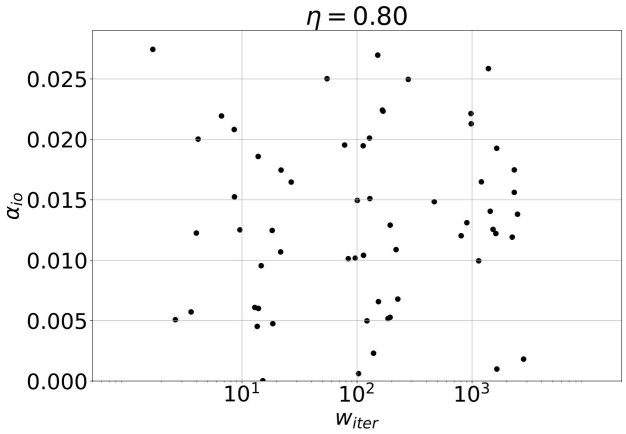




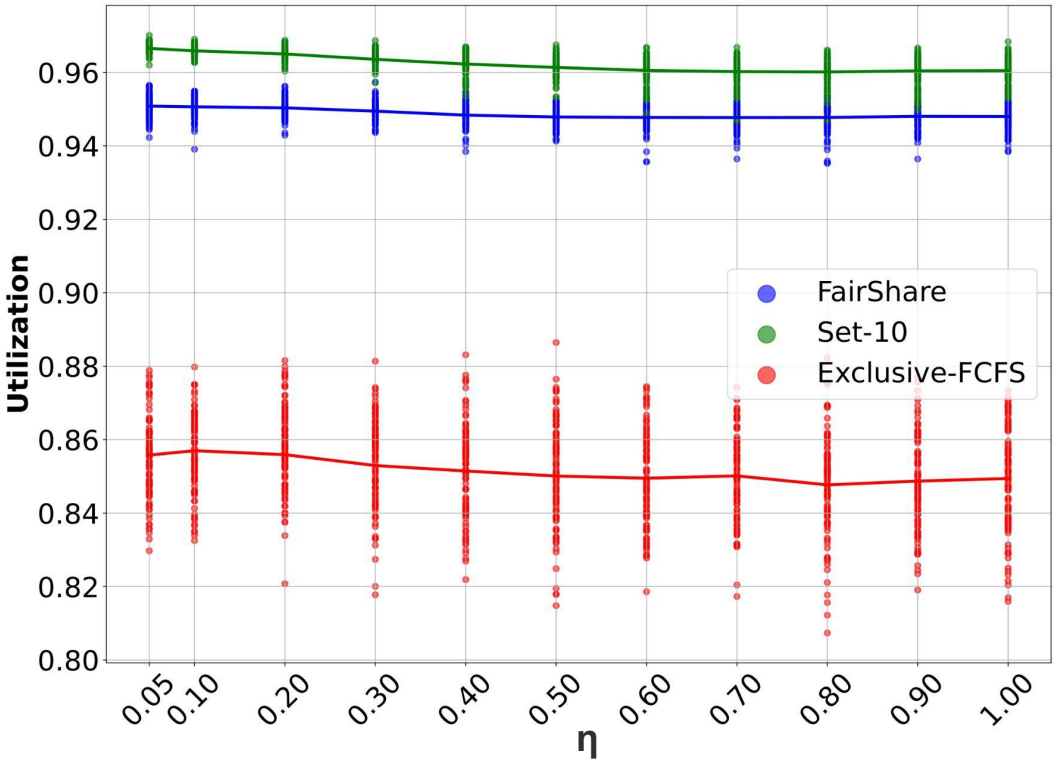
Is it the mapping strategy good in this case?

If the sets are not well-defined

- For various values of η we generate workloads with:
  - Twenty jobs with witer ~ N(10, η10)
  - Twenty jobs with witer ~ N (100, η100)
  - Twenty jobs with witer ~ N (1000, η1000)



If the sets are not well-defined



The variability is handled by moving the applications into different sets of similar orders of magnitude.

Conclusion

- Set-10 is always better than fairshare and exclusive
- **I/O performance improved in up to 25%**
- Omitted results, check our paper https://hal.inria.fr/hal-03648225/
  - Noise in the duration of I/O phases
  - Comparison to other mapping strategies
  - **Set-10 is robust** and performs better (or at least the same) than fairshare
- w_{iter} is a robust metric
  - easy to calculate, lightweight
  - we can adapt it to changes in the application

Practical Applicability

- <u>How to implement I/O sets?</u>
- We believe it should be transparent to applications
  - Intercept all application requests
  - An application agent talks to a centralized scheduler
  - Alternative: we could implement it in the intermediate I/O nodes
- How to enforce **priority-based bandwidth sharing**? Two ideas:
  - Weighted Fair Queuing (WFQ) request scheduling
  - Adapting the number of processes used by the application

Future research directions

- how to detect start and end of **I/O phases**?
- the impact of phase detection on w_{iter} calculation
- how to deal with applications that do not use their share of the bandwidth (access pattern)
- apply IO-Sets to other levels of the I/O stack
  - for example: control access to shared Burst Buffers

IO-SETS: Simple and efficient approaches for I/O bandwidth management

Thanks for your attention!

https://hal.inria.fr/hal-03648225/

RÉGION Nouvelle-Aquitaine

université de BORDEAUX

LaBRI

Inria

ADMIRE

malleable data solutions for HPC

References

- Francieli Boito, Guillaume Pallez, Luan Teylo, Nicolas Vidal. I**O-SETS: Simple and efficient approaches for I/O bandwidth management**. 2022. ⟨hal-03648225v2⟩ (under review in TPDS).

- Francieli Boito, Guillaume Pallez, Luan Teylo. **The role of storage target allocation in applications' I/O performance with BeeGFS.** CLUSTER 2022 - IEEE International Conference on Cluster Computing, Sep 2022, Heidelberg, Germany. ⟨hal-03753813⟩

- Francieli Boito, Antonio Tadeu A. Gomes, Louis Peyrondet, Luan Teylo. **I/O performance of multiscale finite element simulations on HPC environments**. WAMCA 2022 - 13th Workshop on Applications for Multi-Core Architectures, Nov 2022, Bordeaux, France. ⟨hal-03808833⟩

- Alessa Mayer, Luan Teylo, Francieli Boito. **Implementation and Test of a Weighted Fair Queuing (WFQ) I/O Request Scheduler.** [Research Report] RR-9480, Inria. 2022, pp.12. ⟨hal-03758890v3⟩