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

Francieli Boito, Guillaume Pallez, Luan Teylo, Nicolas Vidal

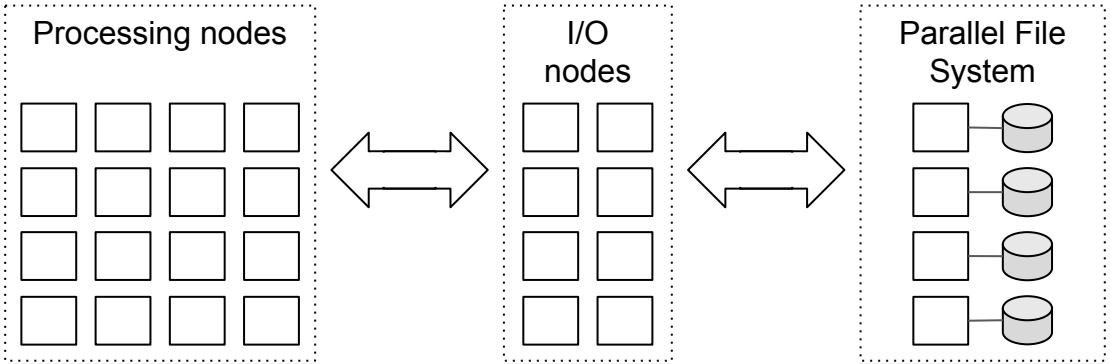
Université de Bordeaux, Inria, LaBRI

Per3S Workshop - June 2022

université de BORDEAUX LaBRI Inria ADMIRE malleable data solutions for HPC

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

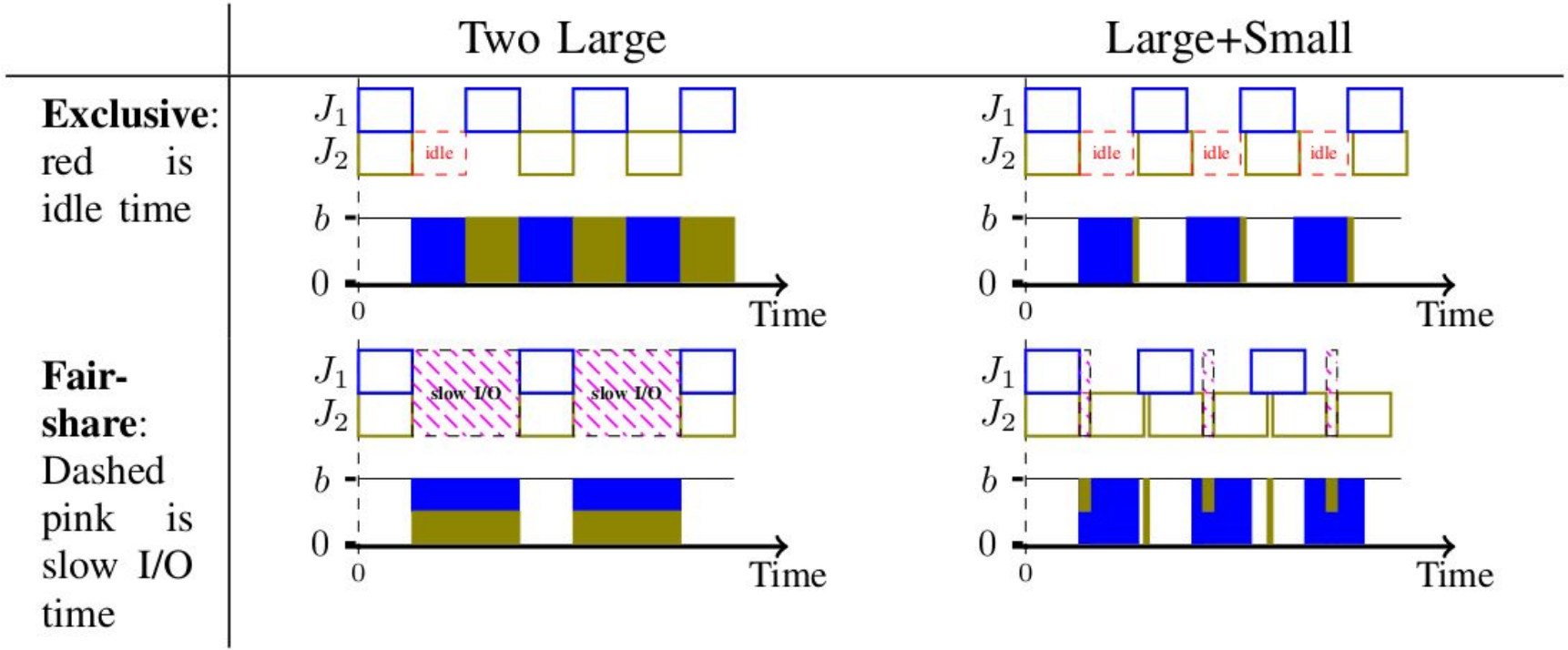


Motivation

- **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

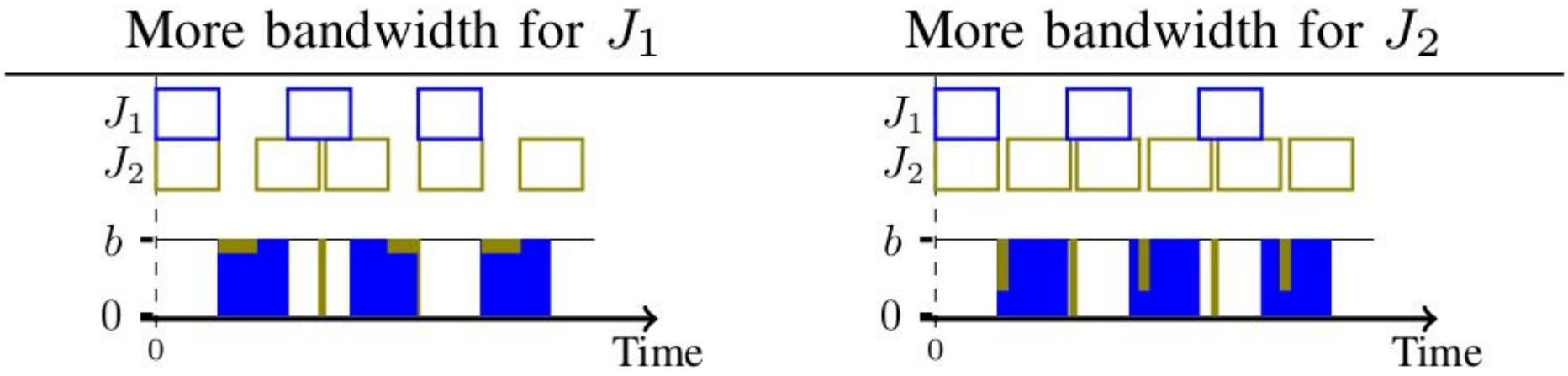
Exclusive vs. Fairshare: an example

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



Exclusive vs. Fairshare: an example

- Two concurrent periodic applications
  - “small” (**J2**) or “large” (**J1**) 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
- Set-10 algorithm in the IO-Sets method:
  - An application is assigned to a set that corresponds to its w_{iter} magnitude order:

$$\pi : A_{\mathrm{id}} \mapsto S_{\lfloor \log_{10} w_{\mathrm{iter}}^{\mathrm{id}} \rceil}$$

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

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

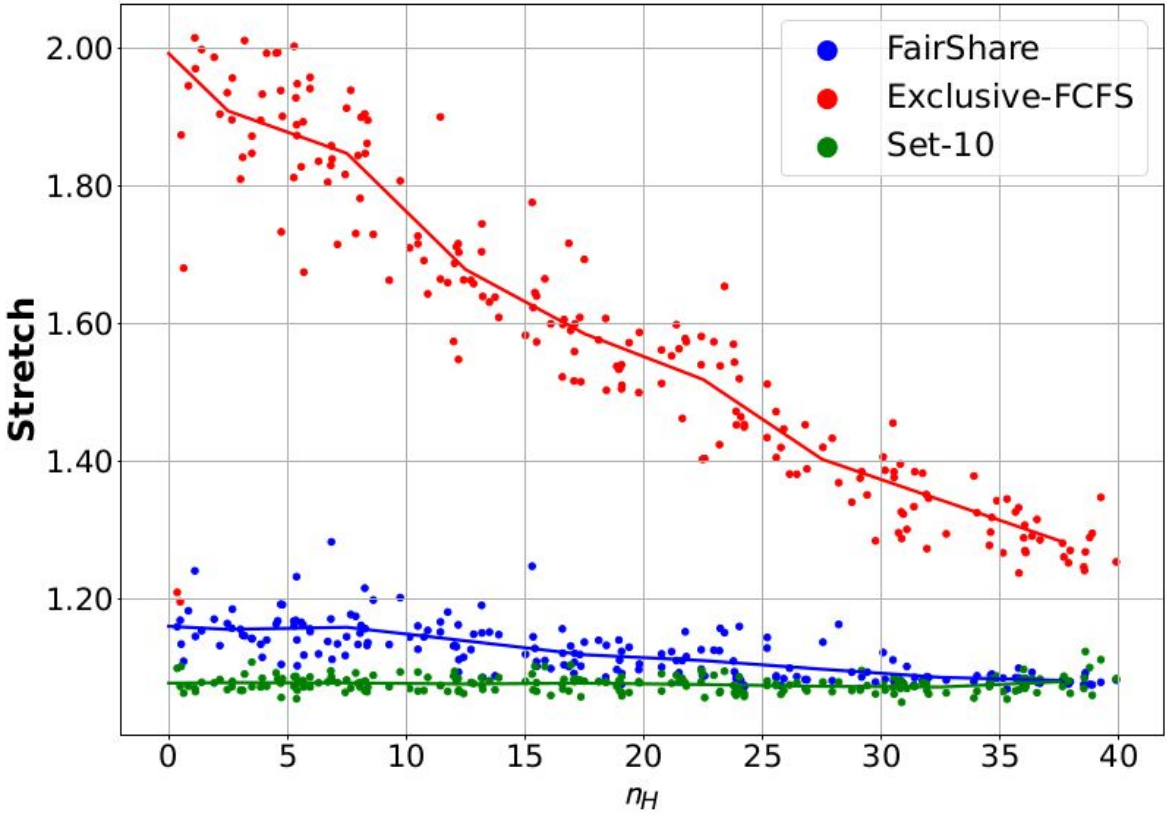
- Applications with the **smallest w_{iter} get the highest priority**, i.e. most of the bandwidth
  - S_1 gets 1/10, S_2 gets 1/100, S_3 gets 1/1000, …

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)
- We vary nH and nL = 40 - nH
- Random amount of data per application, ensuring a total I/O load of 0.8
- Stretch: how many times slower the application runs (compared to running by itself)

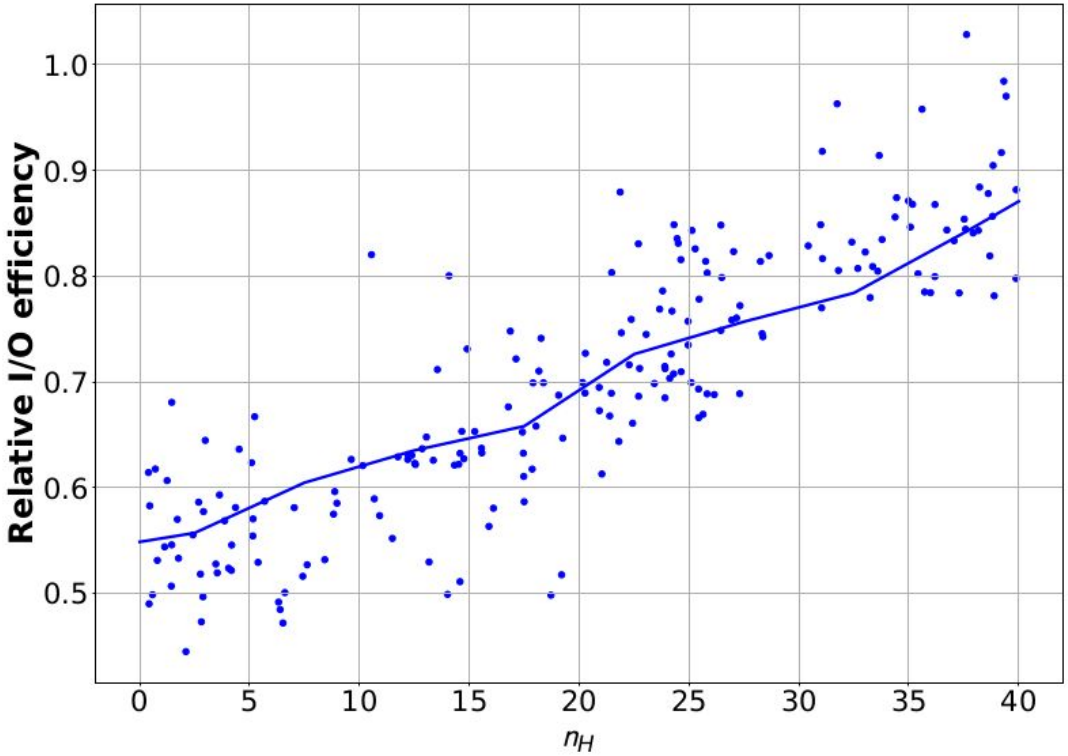
Validation

many low-frequency (long phases) applications



many high-frequency (short phases) applications

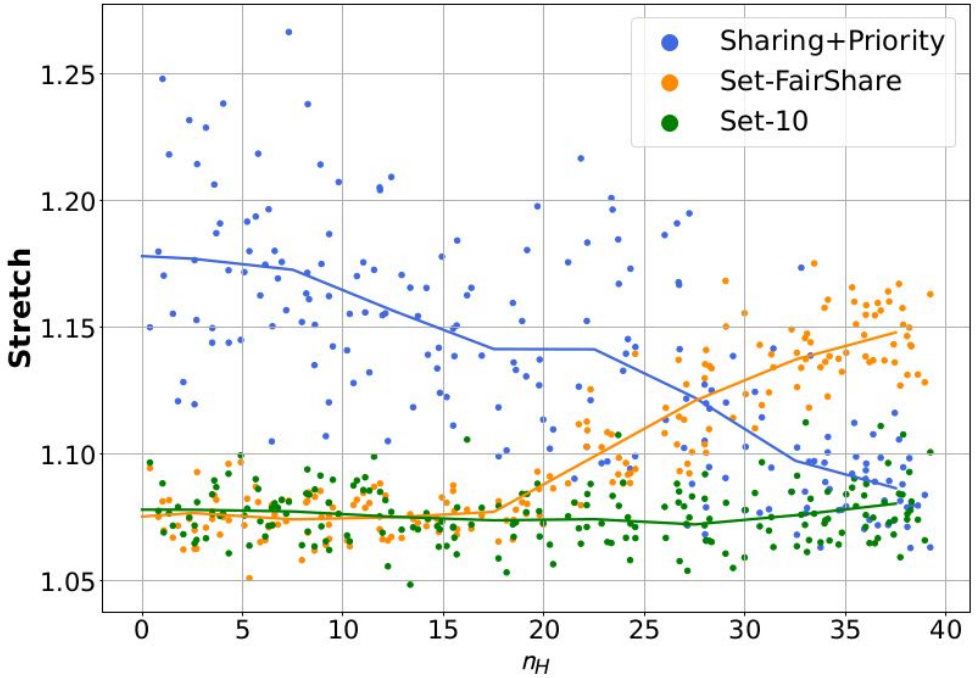
I/O performance impact



$$\texttt{IO-efficiency} = \frac{1}{N}\sum_j \frac{e_j^{iter}.t_{\text{io}}}{(T_{end} - T_{begin}) - e_j^{cpu}}$$

Where do results come from?

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



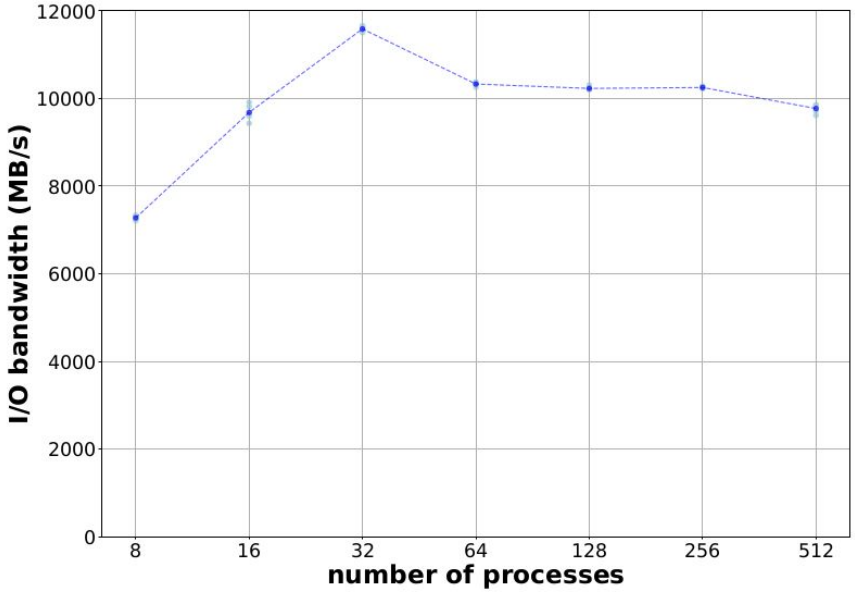
Conclusion so far

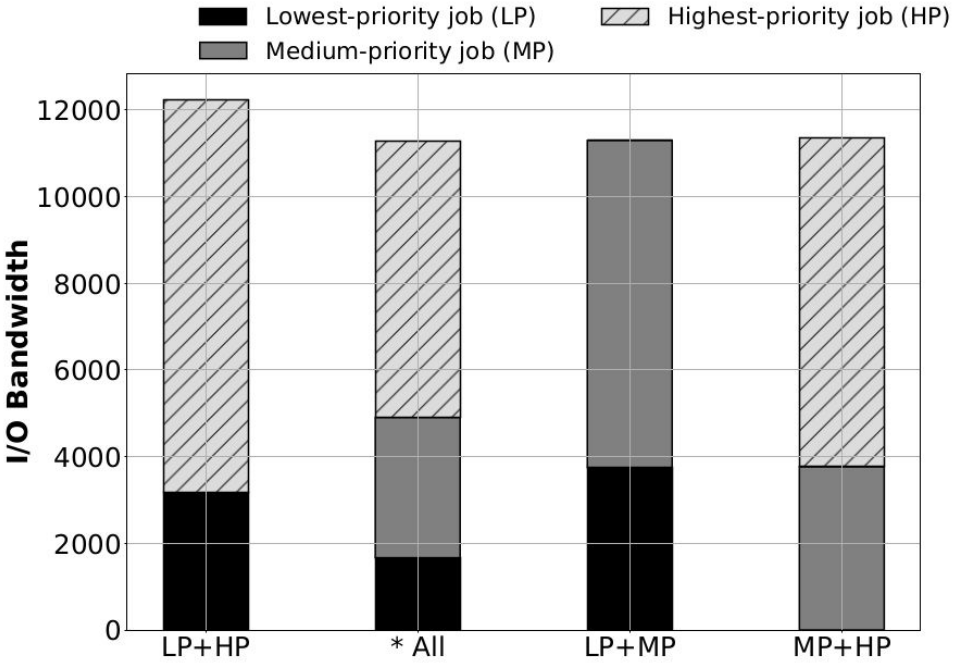
- Set-10 is always better than fair-share and exclusive
- **I/O performance improved in up to 45%**
- Omitted results, check our paper https://hal.inria.fr/hal-03648225/
  - Noise in the duration of I/O phases (aperiodic applications)
  - Variability in w_{iter} so sets are less well defined
  - Addition of a fourth profile but not a new order of magnitude
  - **Set-10 is robust** and performs better (or at least the same) than fair share
- w_{iter} is a robust metric because it is an average
  - easy to calculate, lightweight
  - we can adapt it to changes in the application (for example, EWMA)

Practical Applicability

- How to implement I/O sets?
- 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

Adapting the number of processes used for I/O





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

Thanks for your attention!

If you are interested, please read our paper:

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

université de BORDEAUX LaBRI Inria ADMIRE malleable data solutions for HPC