

Towards Reusability of Autonomic Controllers in High Performance Computing

LIG WAX GLSI

Quentin GUILLOTEAU,^{*} Éric RUTTEN,^{*} Rosa PAGANO,^{*}
Sophie CERF,^{***} Raphael BLEUSE,^{*} Bogdan ROBU^{**}

^{*}Université Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG

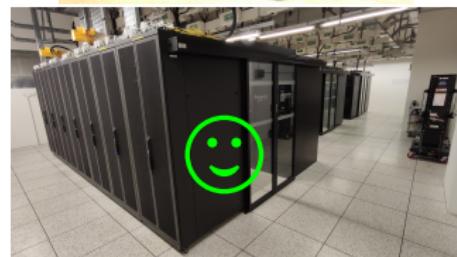
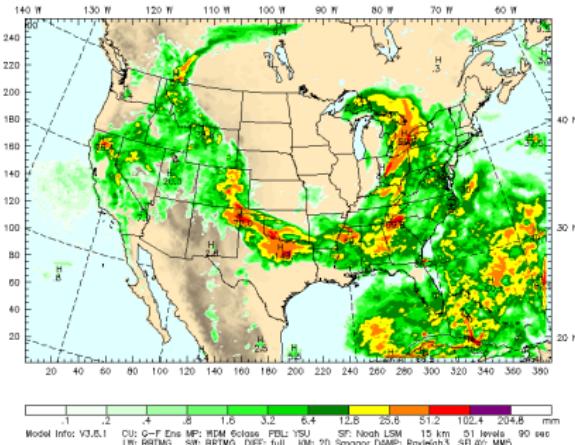
^{**}Université Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab

^{***}Univ. Lille, Inria, CNRS, Centrale Lille, CRISTAL

2023-06-13

Context - High-Performance Computing

15km ARW WRF, NAM-init --- NCAR/MMM
Fcast: 18 h Valid: 00 UTC Mon 12 Jun 23
Total precip. since h 0



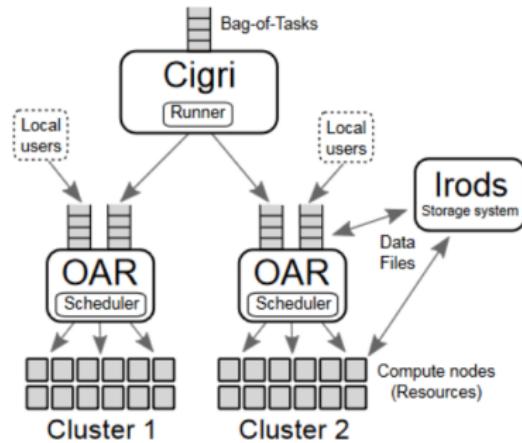
Laptops **too small** ↵ need **several powerful** machines
↪ expensive ↵ shared ↵ **reservation process**

Context

Idle HPC Resources \implies Lost Computing Power \rightsquigarrow How to Harvest ?

One Solution: *CiGri*

- **bag-of-tasks**: many, multi-parametric
- **Best-effort Jobs**: Lowest priority
- **Objective**: Collect grid idle resources



Problem

\nearrow Harvesting \implies \nearrow Perturbations (e.g., I/O) \rightsquigarrow **Trade-off**

\hookrightarrow Unpredictability \implies **runtime management**

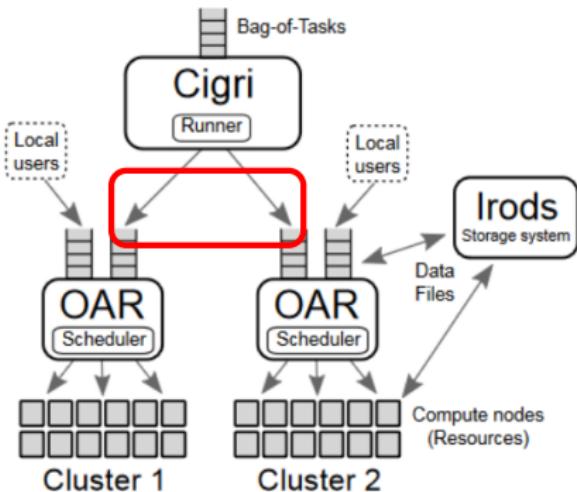
CiGri: Submission Loop (1/2)

Algorithm 1: Current Solution

```

rate = 3;
increase_factor = 1.5;
while tasks not executed in b-o-t do
    if no task running then
        submit rate tasks;
        rate = min(rate ×
                    increase_factor, 100);
    end
    while nb of tasks running > 0
        do
            sleep during 30 sec;
        end
    end

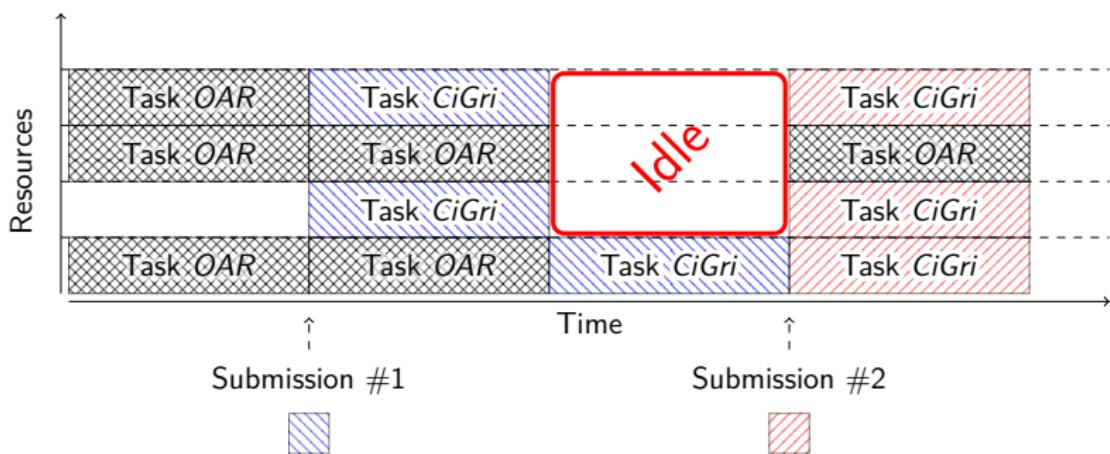
```



CiGri: Submission (2/2)

The Issue

Must wait for termination of the previous submission to submit again

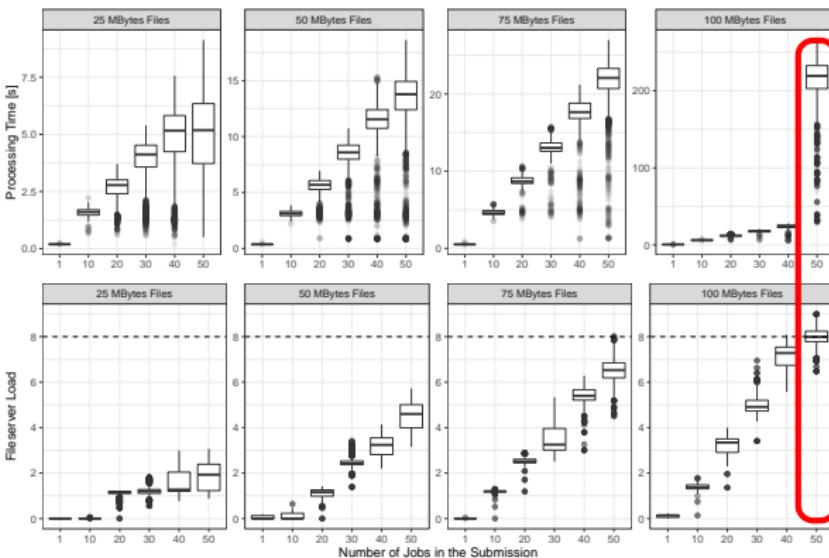


Using all resources = "easy", but ... ↳ jobs are **using shared resources**

Degradation of the File System Performances

↗ Jobs \Rightarrow ↗ I/O \Rightarrow ↗ More delay for users \rightsquigarrow Perturbations

Processing Time and Fileserver Load for different Submissions (number of jobs and filesize)



overload!

Sensor

- `loadavg`
- linear relation
- shows limits of FS
- estimation of perturbations

Runtime management

Autonomic Computing and the MAPE-K Loop

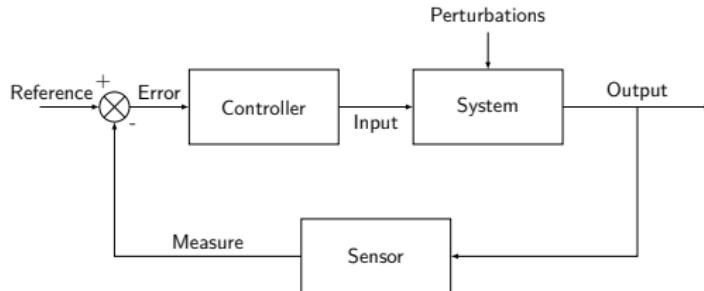
Auto-regulating Systems given high-level objectives

Phases: Monitor \rightsquigarrow Analyse \rightsquigarrow Plan \rightsquigarrow Execute (with Knowledge)

Control Theory (Feedback Control Loop)

Regulate the behaviour of dynamical systems

\hookrightarrow Interpretation of the MAPE-K Loop



Our Global Problem and Objectives

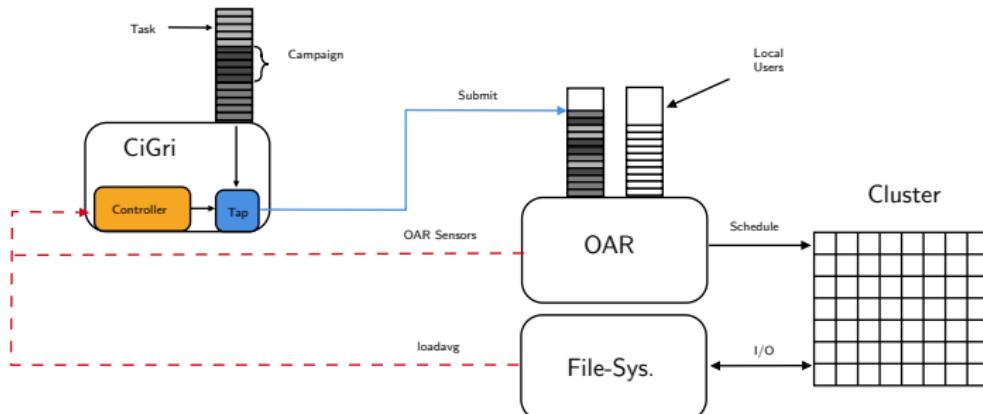
Objective

Harvest Idle Resources in a
non-intrusive way

- max cluster utilization
- min perturbations

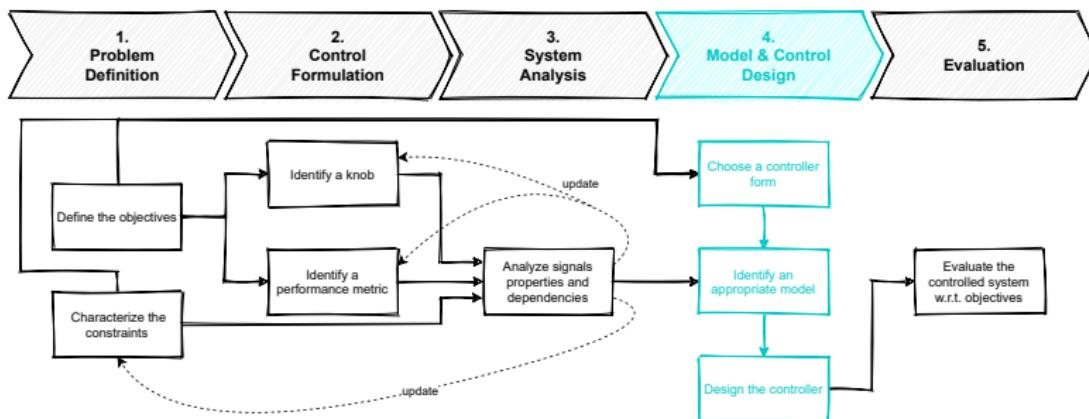
Means

- Instrumentation
 - **Actuator**: #jobs to submit, ...
 - **Sensor**: RJMS WQ, FS Load, ...
- **Controllers** (PID, RST, MFC, ...)
- Experimental Validation



Usual Method (e.g., PID) and Difficulties

→ take into account current state of cluster ↽ **use Control Theory**



However...

Cluster/Grid Administrators are **not** Control Theory experts

→ **Design Cost?** **Setup Cost?** Runtime **Performances?**

Comparison Framework

Three* Controllers

- Proportional-Integral (PI)
- Model-Free (MFC)
- *Adaptive PI*

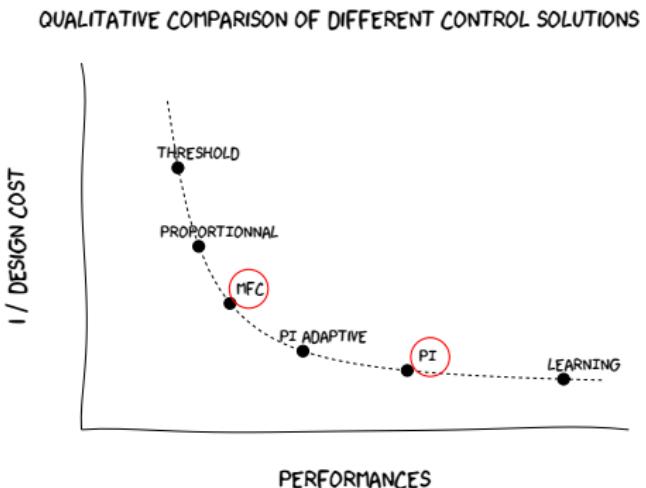
Variations: jobs (I/O, duration)

Reusability Criteria

- Design Time Cost
- Runtime Behavior

Goal

Compare Controllers Reusability: Design Cost vs. Runtime Behavior



1 Introduction & Context

2 Design of Controllers

- Proportional-Integral
- Model-Free Control
- Adaptive Proportional-Integral
- Ease of Design/Setup

3 Experimental Comparison

4 Conclusion & Future Work

PI: What are we looking for

First, a Model ... (i.e., how does the system behave (Open-Loop))

$$\mathbf{y}(k+1) = \sum_{i=0}^k a_i \mathbf{y}(k-i) + \sum_{j=0}^k b_j \mathbf{u}(k-j)$$

... then a (PID) Controller (i.e., the Closed-Loop behavior)

$$\text{Output} = \mathbf{K}_p \times \text{Error}_k + \mathbf{K}_i \times \sum_k \text{Error}_k + \mathbf{K}_d \times (\text{Error}_k - \text{Error}_{k-1})$$

Sensors & Actuators

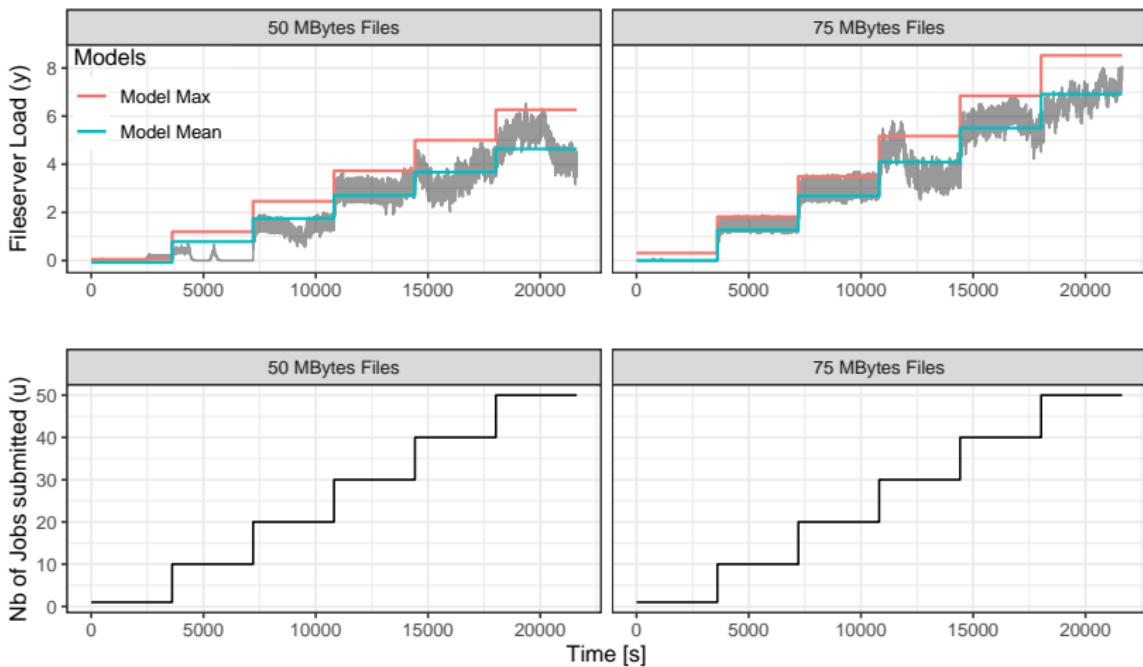
- Actuator: #jobs to sub $\rightsquigarrow \mathbf{u}$
- Sensor: FS Load $\rightsquigarrow \mathbf{y}$
- Error: *Reference – Sensor*

Method

- 1 Open-Loop expe (fixed \mathbf{u})
- 2 Model parameters (a_i, b_j)
- 3 Choice controller behavior (\mathbf{K}_*)

PI: Open-Loop and Identification

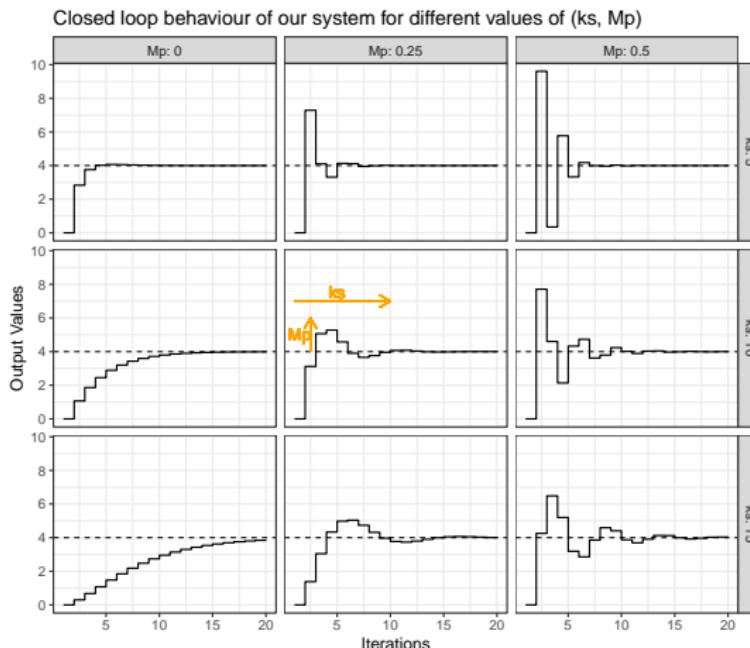
System Identification and (Linear) Model Fitting



$$y_{ss} = \alpha + \beta_1 f + \beta_2 u + \gamma f u$$

PI: Closed-Loop Behavior

Open-Loop Experiments \Rightarrow Model (1st order) $y(k + 1) = ay(k) + bu(k)$ \Rightarrow Controller Gains $K_p, K_i, K_d,$



Controller Gains are ...
functions of the model and

- k_s : max **time** to steady state
- M_p : max **overshoot** allowed

Non-Intrusive Harvesting

- no overshoot
- but "fast" response

1 Introduction & Context

2 Design of Controllers

- Proportional-Integral
- **Model-Free Control**
- Adaptive Proportional-Integral
- Ease of Design/Setup

3 Experimental Comparison

4 Conclusion & Future Work

What is Model-Free Control ? [Fliess & Join]

Model-Free Control

- Introduces *intelligent* Controllers (*iPID*)
- Easier to tune than PI
- Adapt to the plant/system (F)
- can be equivalent to PI

$$\begin{cases} \hat{F}_k &= \frac{y_k - y_{k-1}}{\Delta t} - \alpha \times u_k \\ u_{k+1} &= \frac{-\hat{F}_k - \dot{y}_k^* + K_p \times e_k}{\alpha} \end{cases}$$

- y_k : Load of File System
- u_k : #jobs *CiGri*
- \dot{y}_k^* : Derivative of ref. value

- \hat{F}_k : Estimation of the model
- α : **non-physical cst parameter**
- K_p : **Gain of the controller**

Empirical Choice of Parameters

α such that $\frac{y_k - y_{k-1}}{\Delta t}$ and $\alpha \times u_k$ have same order of magnitude

1 Introduction & Context

2 Design of Controllers

- Proportional-Integral
- **Model-Free Control**
- Adaptive Proportional-Integral
- Ease of Design/Setup

3 Experimental Comparison

4 Conclusion & Future Work

Adaptive Proportional-Integral Controller

Goal

Dynamically estimate model params: $\mathbf{y}(k + 1) = a \mathbf{y}(k) + b \mathbf{u}(k)$

Intuition: LMS on the model params between model pred. vs reality

$$\hat{b}(k + 1) = \hat{b}(k) + V(k + 1) \times \mathbf{u}(k) \times \frac{\varepsilon(k + 1)}{1 + \alpha|\varepsilon(k + 1)|}$$
$$\varepsilon(k + 1) = \mathbf{y}(k + 1) - a\mathbf{y}(k) - \hat{b}(k)\mathbf{u}(k)$$

Based on a PI!

A lot of parameters...

- initial conditions ? $(\hat{b}(0), V(0), \dots)$
- forgetting factor ?
- robust parameter α ? stable ? overshoot ? converges ?

1 Introduction & Context

2 Design of Controllers

- Proportional-Integral
- Model-Free Control
- Adaptive Proportional-Integral
- Ease of Design/Setup

3 Experimental Comparison

4 Conclusion & Future Work

Ease of Design/Setup

	PI	MFC	aPI
Nb Parameters	:)	:-)	:(:(
Identification	:)	:-)	:(
Guarantees	:-)	:(:-)
Explainability	:-)	:(:-)
Adaptability	:)	:-) / :(:-)

→ Which controller to choose?

1 Introduction & Context

2 Design of Controllers

- Proportional-Integral
- Model-Free Control
- Adaptive Proportional-Integral
- Ease of Design/Setup

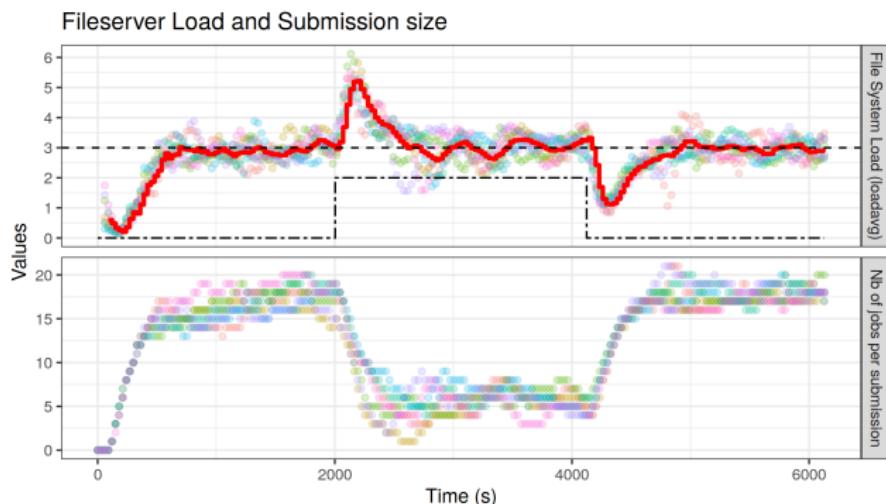
3 Experimental Comparison

4 Conclusion & Future Work

Experimental Setup

Experimental Setup

- Experiments done on Grid'5000
- Emulation of a 100 node cluster
- 2 Intel Xeon E5-2630 v3
- CiGri jobs: sleep + write

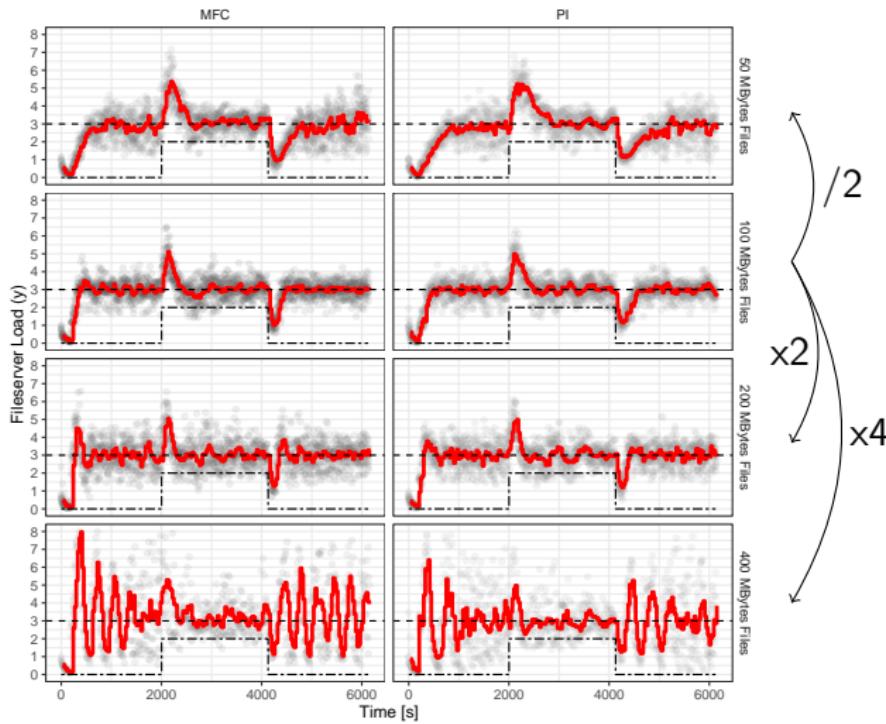


Synthetic Load

- Pure step
- Observe the ctr behavior:
 - response
 - oscillations

Variation in I/O

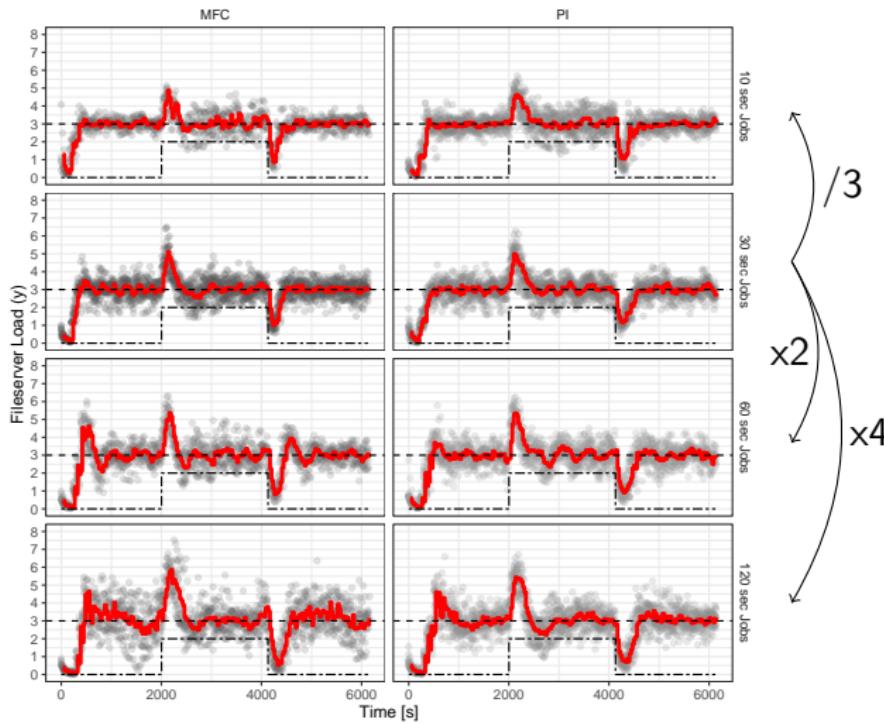
Comparison between the MFC and PI with variations in the I/O impact of jobs



- \simeq behavior
- MFC faster but more aggressive
- PI less variations for larger I/O

Variation in Execution Time

Comparison between the MFC and PI with variations in the Execution Times of jobs

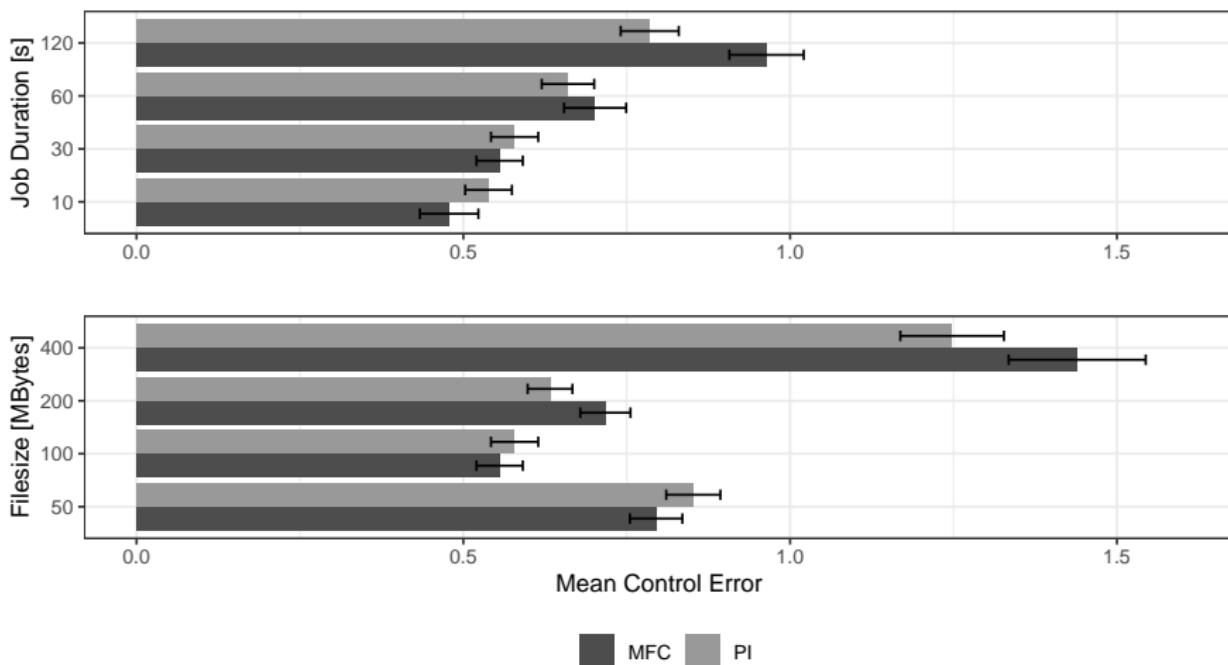


- \simeq behavior
- MFC faster but more aggressive
- Job duration variations have less impact on control quality than the I/O quantity

Performances Comparison

Comparison of the Mean Control Errors for the Controllers with different Variations

99% confidence intervals



1 Introduction & Context

2 Design of Controllers

- Proportional-Integral
- Model-Free Control
- Adaptive Proportional-Integral
- Ease of Design/Setup

3 Experimental Comparison

4 Conclusion & Future Work

Conclusion & Future Work

Reminder of the Objective

Investigate the **Reusability** of Autonomic Controllers in HPC

Results

Compared 3* Controllers: (PI, MFC, aPI*) on I/O and job dur.

Variations

- MFC has **smaller design cost**, but PI has **behavior guarantees**
- \simeq performances for both controllers (MFC slightly worse)
 - ↪ **MFC seems more reusable than PI**

Future Work

- Run experiments for PI Adaptive
- Investigate more variations dimensions (e.g., type of FS, change in ~~bandwidth~~)