# Towards Reusability of Autonomic Controllers in High Performance Computing

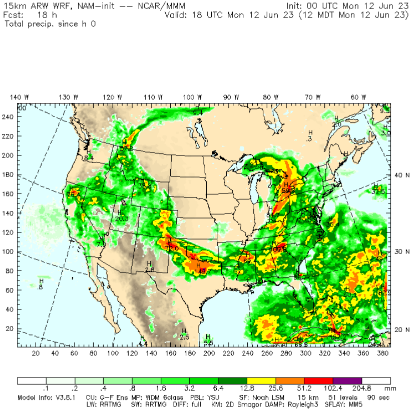
LIG WAX GLSI

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




Laptops **too small** ⇝ need **several powerful** machines

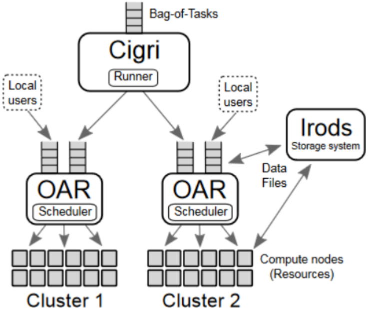
↪ expensive ⇝ shared ⇝ **reservation process**

# Context

Idle HPC Resources $\Longrightarrow$ 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

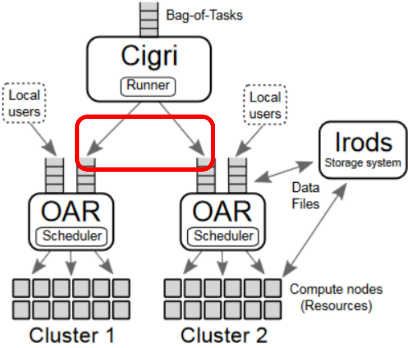
$\nearrow$ Harvesting $\Longrightarrow$ $\nearrow$ Perturbations (e.g., I/O) $\rightsquigarrow$ **Trade-off**

$\hookrightarrow$ Unpredictability $\Longrightarrow$ **runtime management**

# *CiGri*: Submission Loop (1/2)

**Algorithm 1:** Current Solution

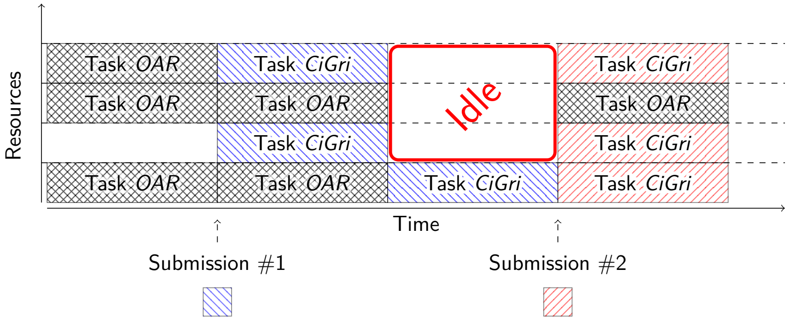
$rate = 3$;
$increase\_factor = 1.5$;
**while** *tasks not executed in b-o-t* **do**
  **if** *no task running* **then**
    submit *rate* tasks;
    $rate = \min(rate \times increase\_factor, 100)$;
  **end**
  **while** *nb of tasks running > 0* **do**
    sleep during 30 sec;
  **end**
**end**



# *CiGri*: Submission (2/2)

## The Issue

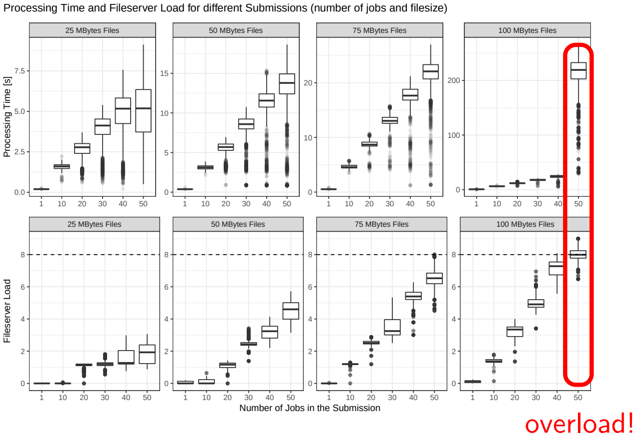
**Must wait for termination** of the previous submission to submit again




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

# Degradation of the File System Performances

$\nearrow$ Jobs $\Longrightarrow$ $\nearrow$ I/O $\Longrightarrow$ $\nearrow$ More delay for users $\rightsquigarrow$ **Perturbations**




## Sensor

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

# Runtime management

## Autonomic Computing and the MAPE-K Loop

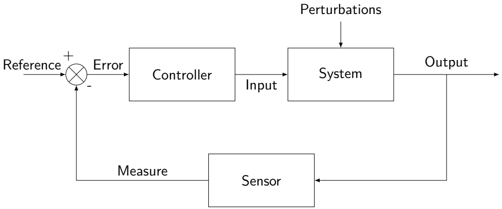
**Auto-regulating** Systems given **high-level objectives**

Phases: **M**onitor ⇝ **A**nalyse ⇝ **P**lan ⇝ **E**xecute (with **K**nowledge)

## Control Theory (Feedback Control Loop)

Regulate the behaviour of dynamical systems

↪ Interpretation of the MAPE-K Loop



# Our Global Problem and Objectives

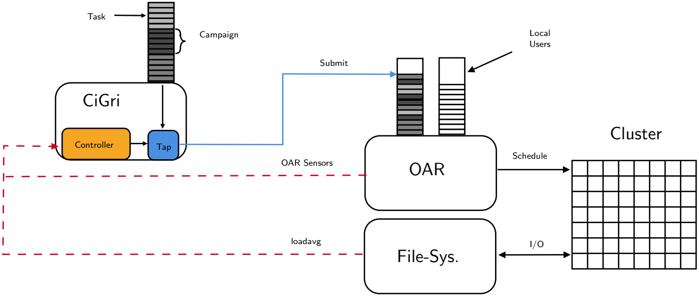
## Objective

Harvest Idle Resources in a **non-intrusive** way

- max cluster utilization
- min perturbations

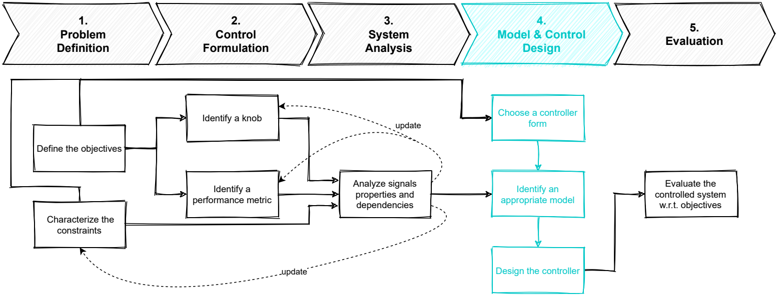
## Means

- 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

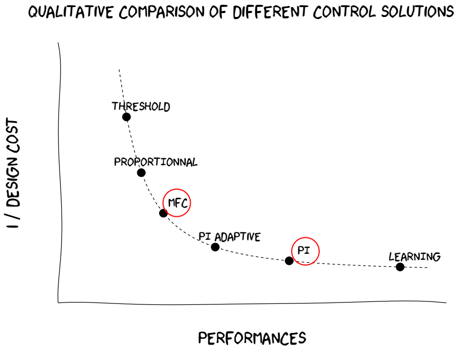
## Three* Controllers

- 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

# 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)

$$Output = \mathbf{K}_p \times Error_k + \mathbf{K}_i \times \sum_{k} Error_k + \mathbf{K}_d \times (Error_k - Error_{k-1})$$

## Sensors & Actuators

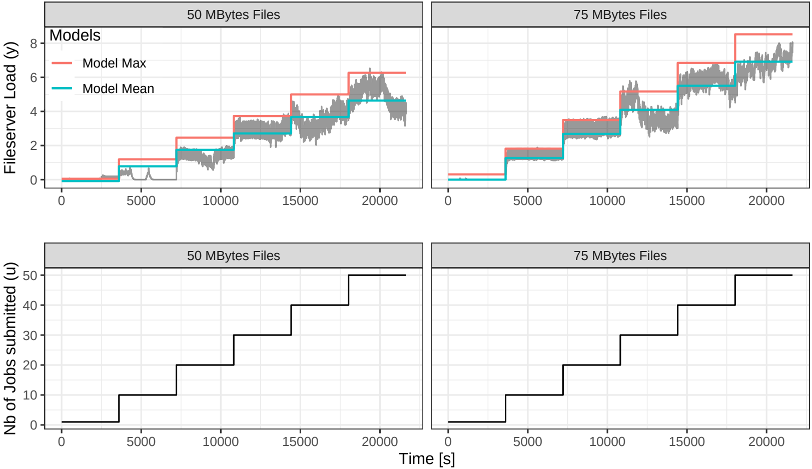
- Actuator: #jobs to sub $\rightsquigarrow$ $\mathbf{u}$
- Sensor: FS Load $\rightsquigarrow$ $\mathbf{y}$
- Error: $Reference - Sensor$

## 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

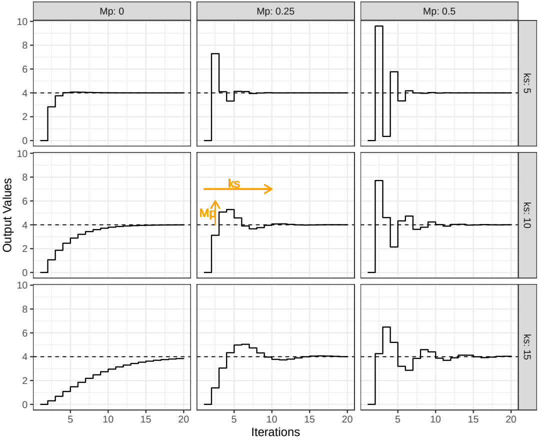
System Identification and (Linear) Model Fitting




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

# PI: Closed-Loop Behavior

Open-Loop Experiments $\Longrightarrow$ Model (1st order) $\mathbf{y}(k+1) = a\mathbf{y}(k) + b\mathbf{u}(k)$ $\Longrightarrow$ Controller Gains $\mathbf{K}_p, \mathbf{K}_i, \mathbf{K}_d,$




## 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

# 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

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

- $y_k$: Load of File System
- $u_k$: #jobs *CiGri*
- $\dot{y}^\star_k$: Derivative of ref. value
- $\hat{F}_k$: Estimation of the model
- $\alpha$: **non-physical cst parameter**
- $\mathbf{K}_p$: **Gain of the controller**

## Empirical Choice of Parameters

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

# 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 ? ($\hat{b}(0)$, $V(0)$, ...)
- forgetting factor ?
- robust parameter $\alpha$ ? stable ? overshoot ? converges ?

# Ease of Design/Setup

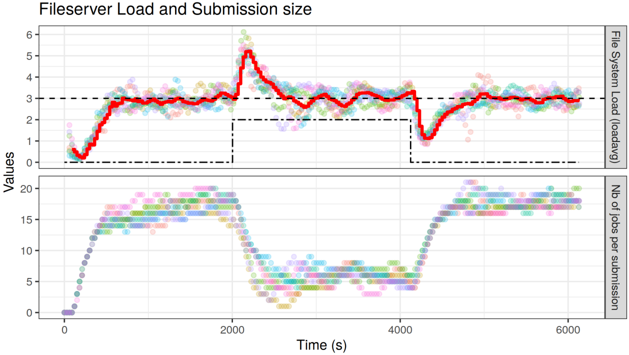
| | PI | MFC | aPI |
|---|---|---|---|
| Nb Parameters | ☹ | ☺ | ☹☹ |
| Identification | ☹ | ☺ | ☹ |
| Guarantees | ☺ | ☹ | ☺ |
| Explainability | ☺ | ☹ | ☺ |
| Adaptability | ☹ | ☺/☹ | ☺ |

↪ Which controller to choose?

# 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

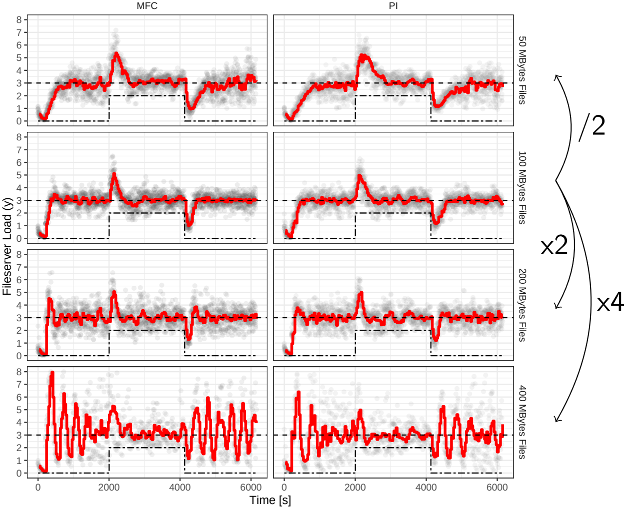



## Synthetic Load

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

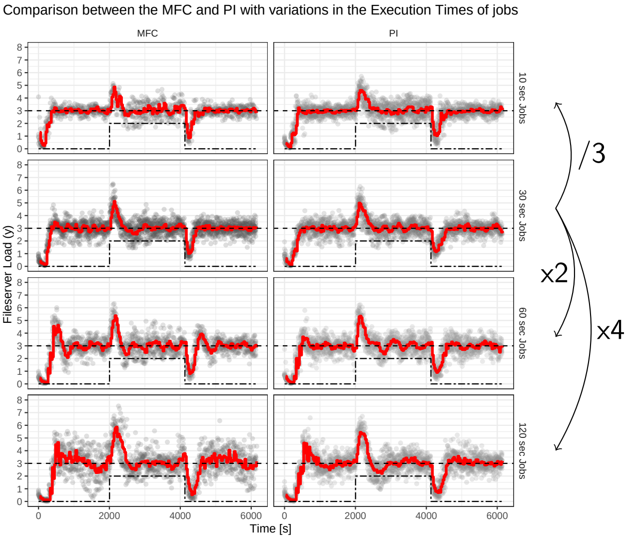
# Variation in I/O

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



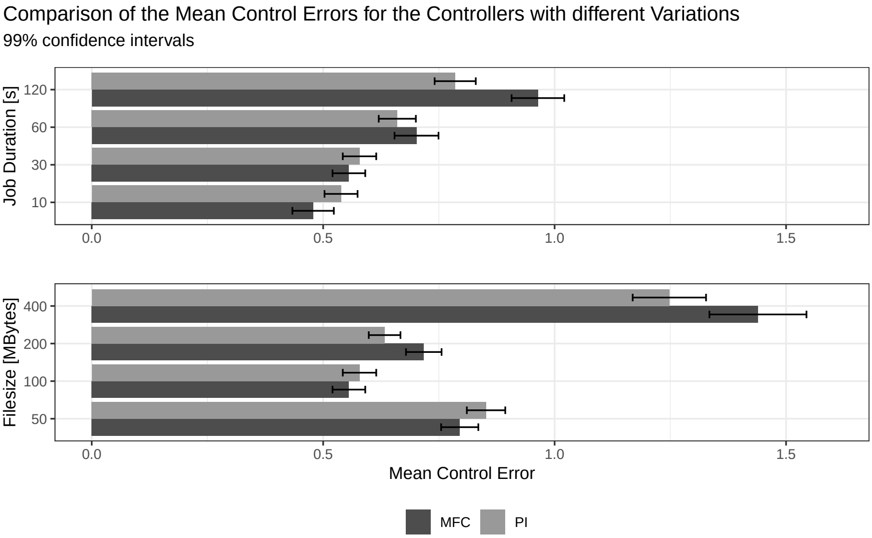

- $\simeq$ behavior
- MFC faster but more aggressive
- PI less variations for larger I/O

# Variation in Execution Time




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

# Performances Comparison



# 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 & job dur. Variations

- MFC has **smaller design cost**, but PI has **behavior guarantees**
- $\simeq$ performances for both controllers (MFC slightly worse)

$\hookrightarrow$ **MFC seems more reusable than PI**

## Future Work

- Run experiments for PI Adaptive
- Investigate more variations dimensions (e.g., type of FS, HW)