Model-Free Control for Resource Harvesting in Computing Grids

CCTA 2022, Trieste

Quentin GUILLOTEAU,[*] Bogdan ROBU,[**] Cédric JOIN,[†] Michel FLIESS,[‡]
Éric RUTTEN,[*] Olivier RICHARD[*]

[*]Université Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
[**]Université Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab
[†]CRAN (CNRS, UMR 7039), Université de Lorraine
[‡]LIX (CNRS, UMR 7161), École polytechnique

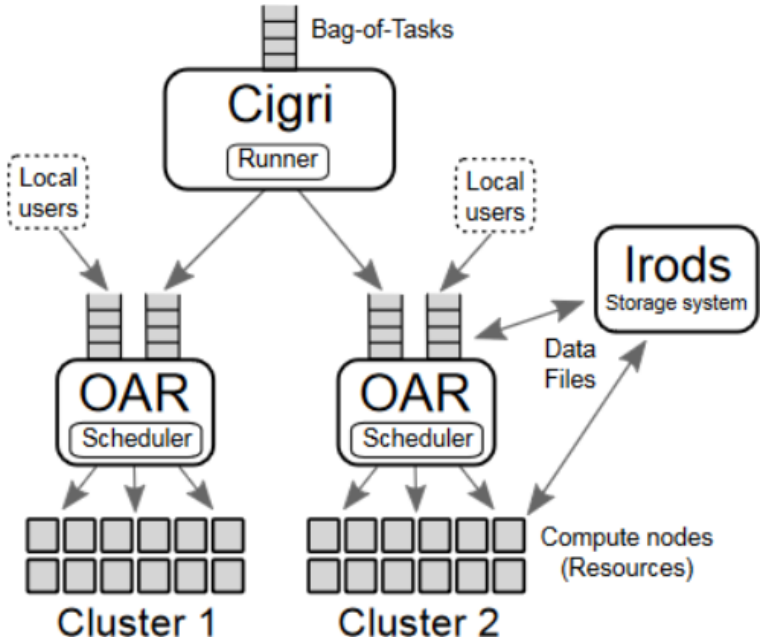
2022-08-23

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**

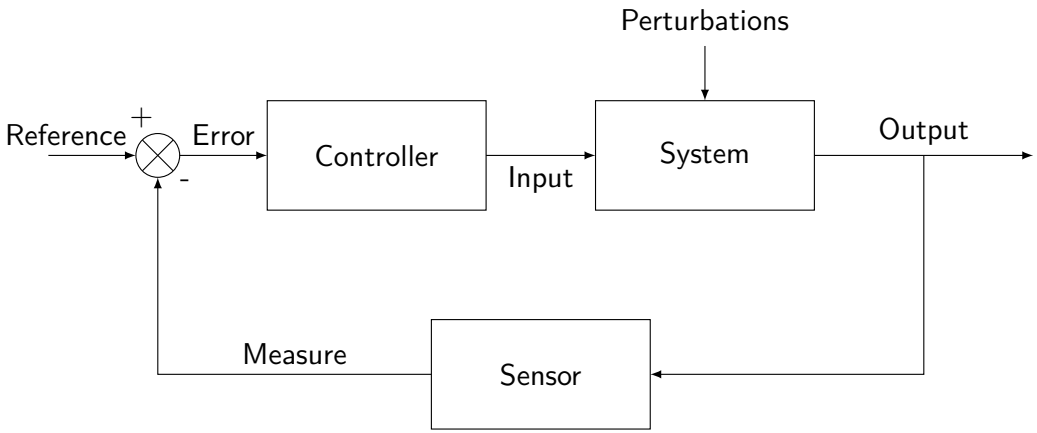
Runtime management

Autonomic Computing and the MAPE-K Loop

Auto-regulating Systems given **high-level objectives**
Phases: **M**onitor $\rightsquigarrow$ **A**nalyse $\rightsquigarrow$ **P**lan $\rightsquigarrow$ **E**xecute (with **K**nowledge)

Control Theory (Feedback Control Loop)

Regulate the behaviour of dynamical systems
$\hookrightarrow$ Interpretation of the MAPE-K Loop



1 Introduction & Context

2 The Problem to Tackle

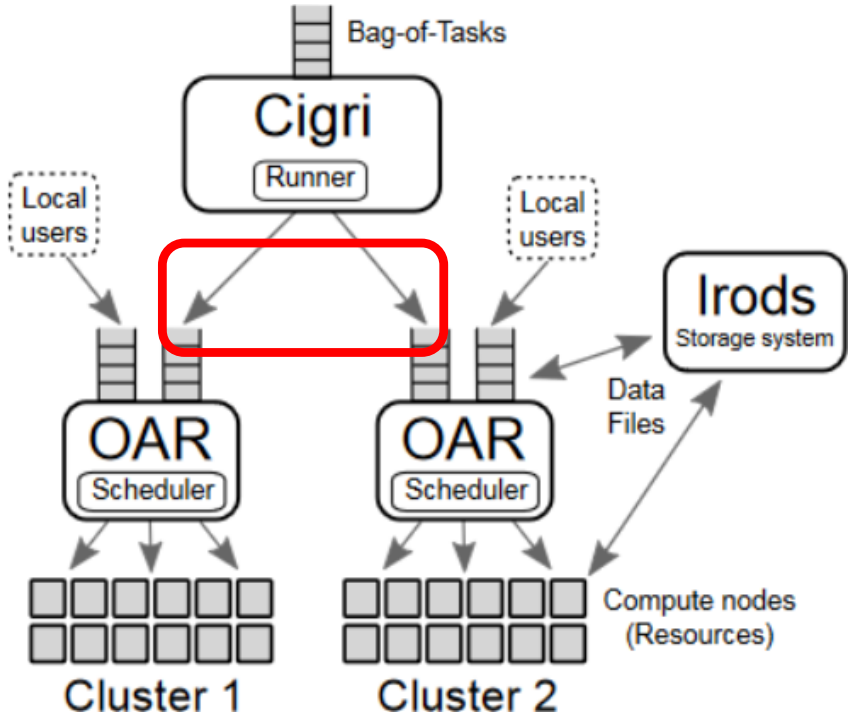
3 Experimental Validation

4 Conclusion & Perspectives

CiGri: Submission Loop (1/2)

Algorithm 1: Current Solution

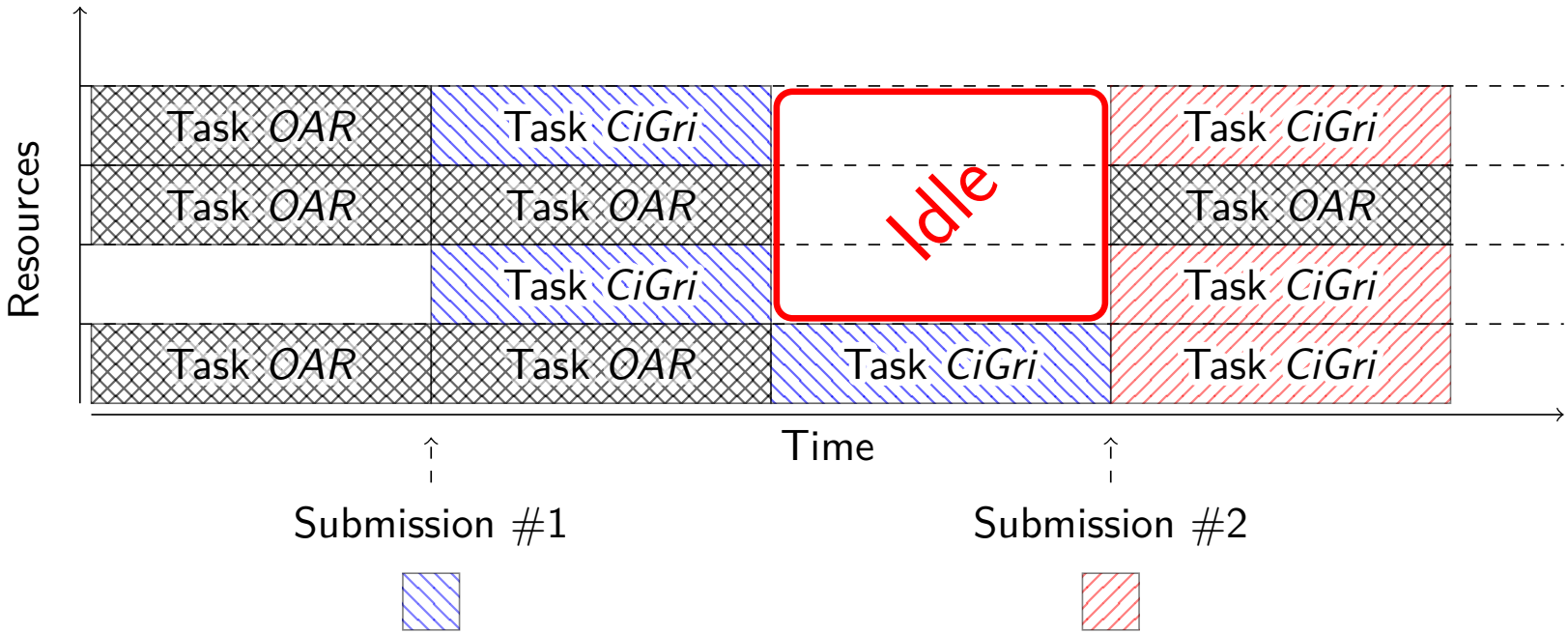
```
rate = 3;
increase_factor = 1.5;
while tasks not executed in b-o-t do
    if no task running then
        submit rate tasks;
        rate = min(rate ×
          increase_factor, 100);
    end
    while nb of tasks running > 0
     do
        sleep during 30 sec;
    end
end
```



CiGri: Submission (2/2)

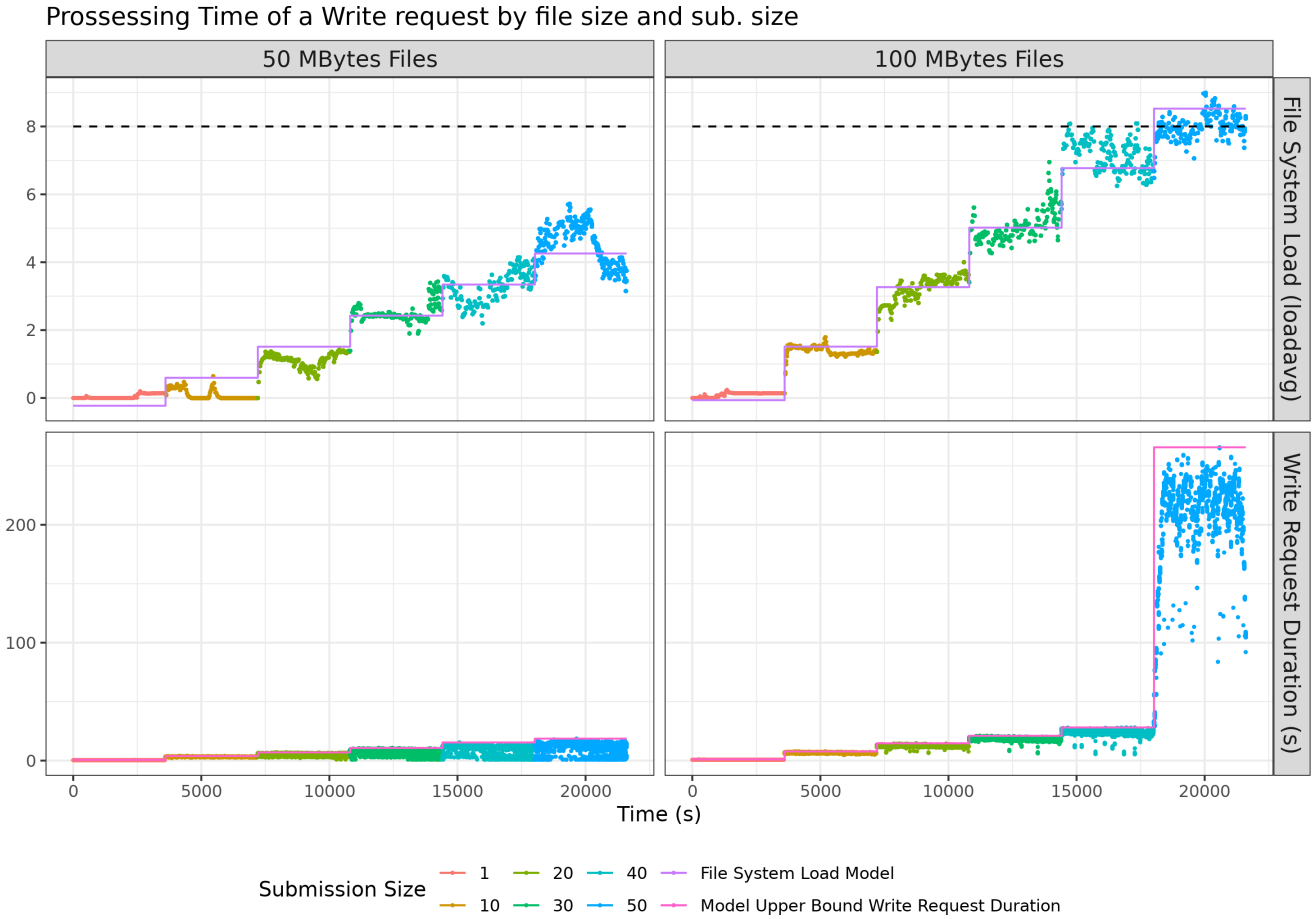
The Issue

Must wait for termination of the previous submission to submit again
↪ reduce overload but introduce **under-utilisation** of the resources



Degradation of the File System Performances

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



Sensor

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

Our Global Problem and Objectives

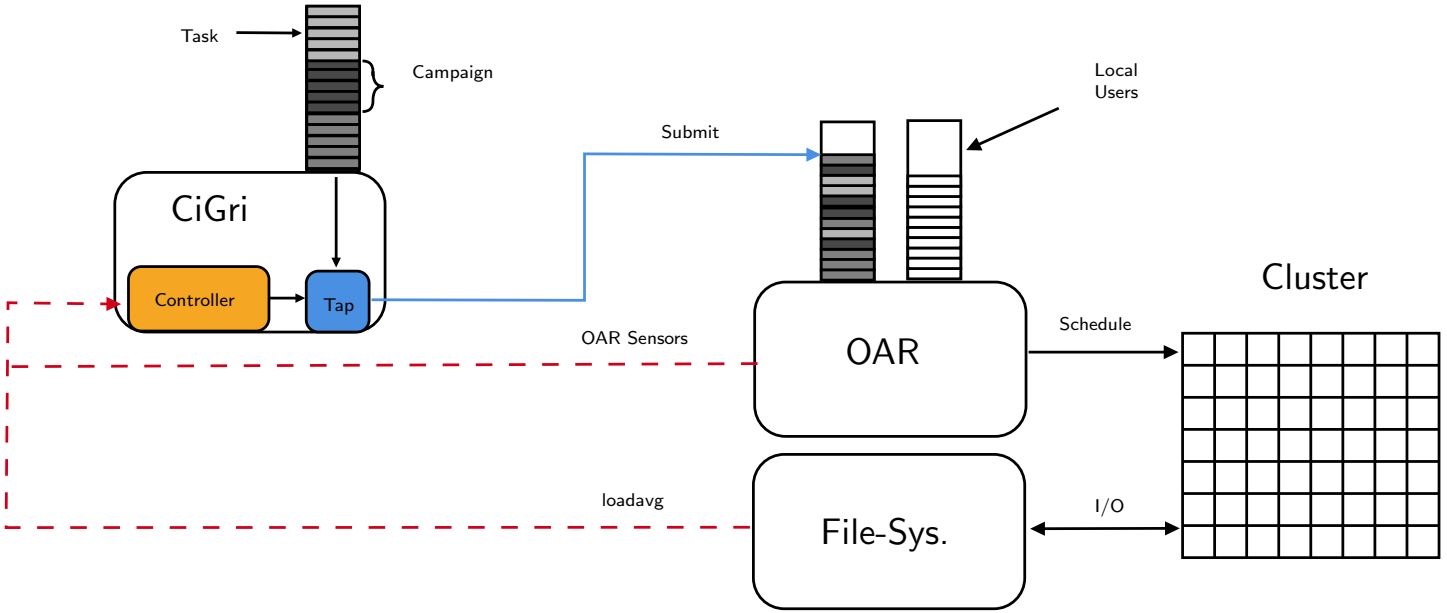
Objective

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

- max cluster utilization
- min perturbations

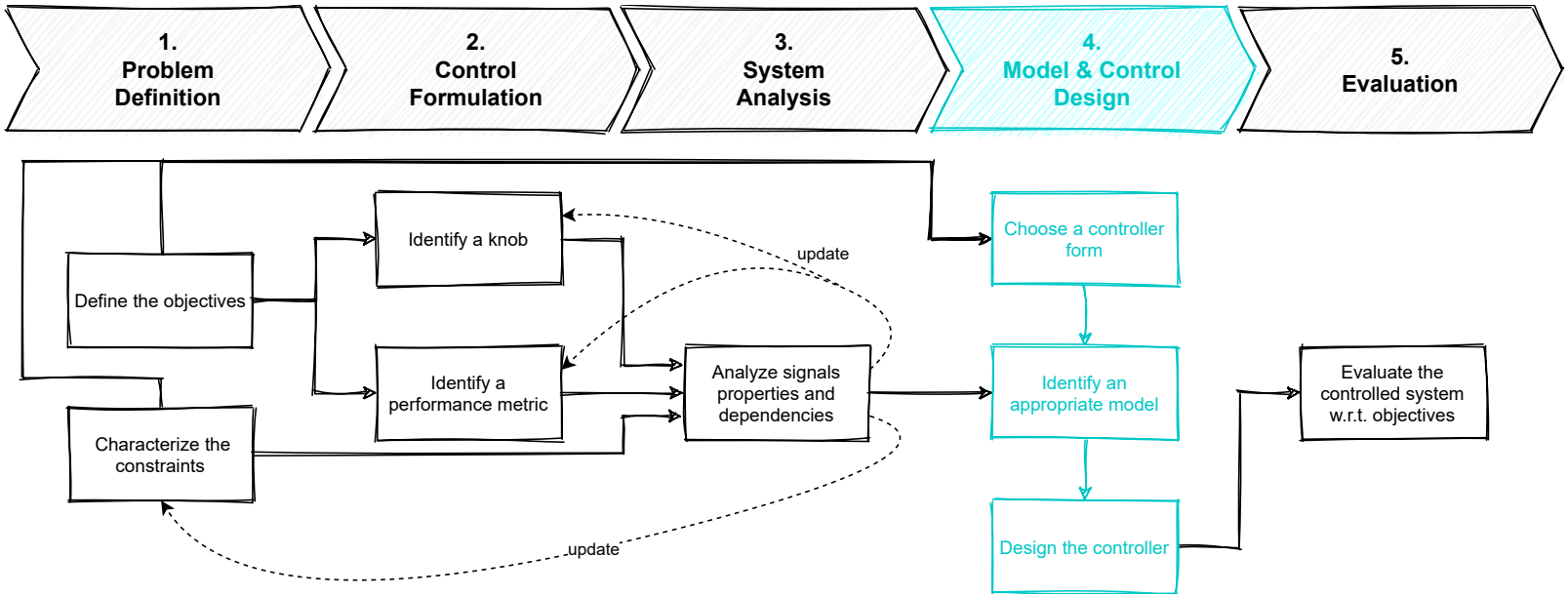
Means

- 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 experts

↪ Use a **Model-Free** approach

What is Model-Free Control ?

Model-Free Control

- Introduces *intelligent* Controllers (*iPID*)
- **Easier to tune**
- **Adapt to the plant**

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

- 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
- α: **non-physical cst parameter**
- K_p: **Gain of the controller**

Problem Objective

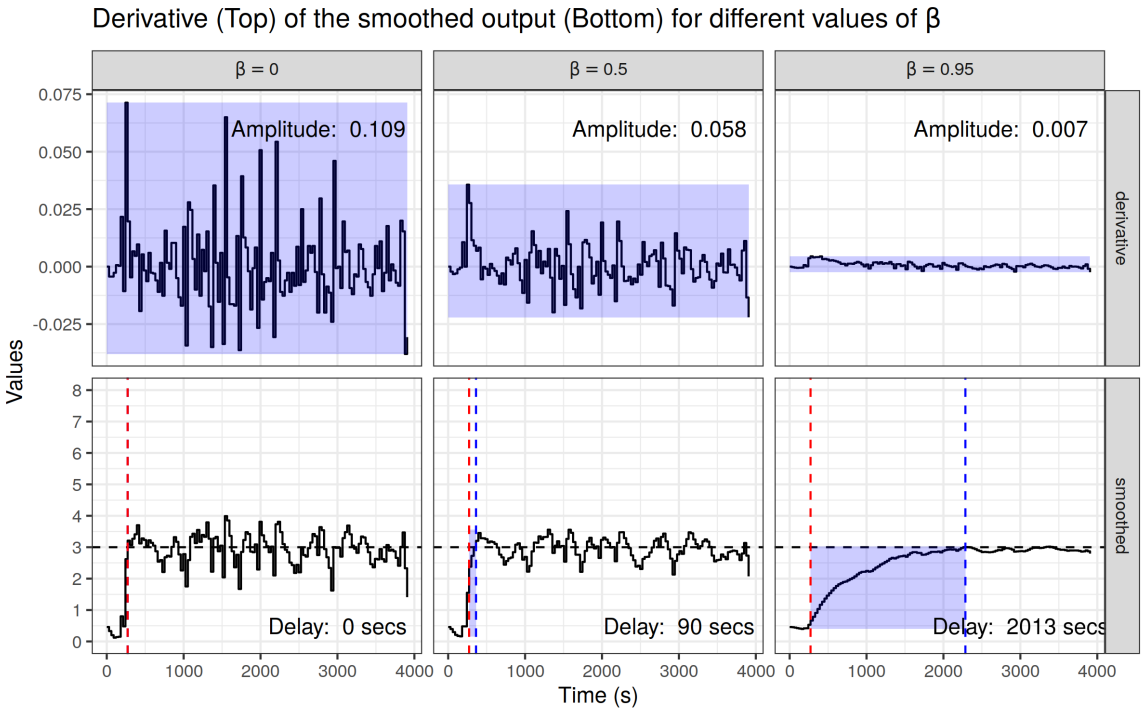
Apply Model-Free Control (*iP*) to the injection of *CiGri* jobs to **regulate the load of the FS**

Dealing with Noisy Sensor

Sensor Noise

Need to compute derivative **but** noisy sensor $\Longrightarrow$ noisy derivative

$\hookrightarrow$ **Filter**/Smooth the output y: $s_k = \beta \times s_{k-1} + (1 - \beta) \times y_k$



Tradeoff

↗ Smooth $\Longrightarrow$ ↗ Delay

↘ Smooth $\Longrightarrow$ ↗ Noise

$\hookrightarrow \beta = ?$

$\hookrightarrow \hat{F}_k = \frac{s_k - s_{k-1}}{\Delta t} - \alpha \times u_k$

Choice of Parameters

Smoothing Factor β

If response to a step y: $s_k = \beta \times s_{k-1} + (1-\beta) \times y = y \times \left(1 - \beta^k\right)$

$\hookrightarrow$ How many iterations (k) to reach p % of the step (y) ?

$\hookrightarrow$ $k = 2, p = 0.95$ seems good $\rightsquigarrow$ $\beta = 0.22$

α

Theory tells us: $\alpha \times u$ same order of magnitude as $\dot{y}$ $(\hat{F} = \dot{y} - \alpha \times u)$

Amplitude $\dot{y} \simeq 0.1 \times (1-\beta)$ and $u \in [0, r_{\max}]$ $\rightsquigarrow$ $\alpha = \frac{0.1 \times (1-\beta)}{r_{\max}} = 0.008$

Proportional Gain K_p

$K_p \times e$ same order of magnitude as $\hat{F}$ $\rightsquigarrow$ $K_p = 0.1 \times (1-\beta) = 0.078$

1 Introduction & Context

2 The Problem to Tackle

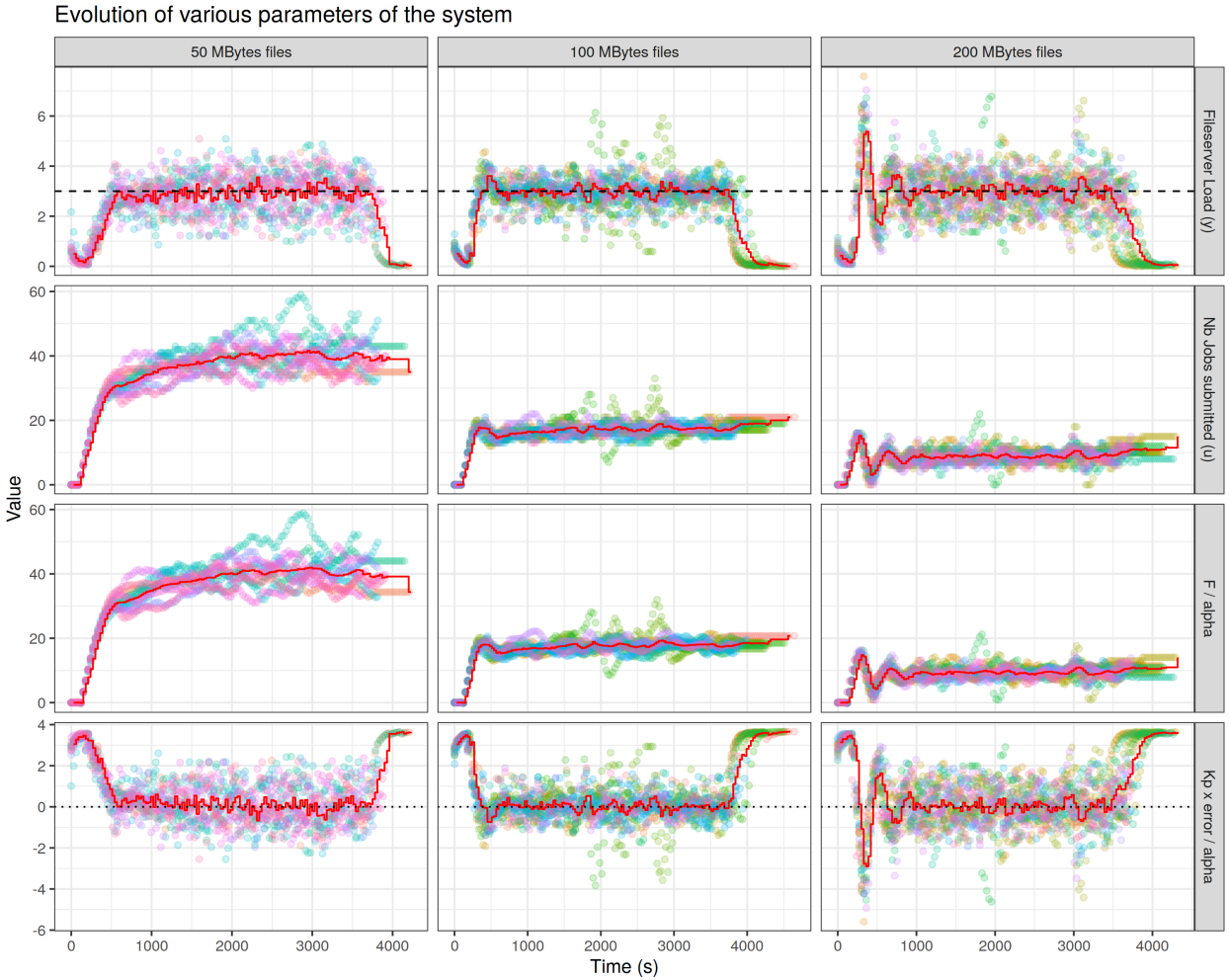
3 Experimental Validation

4 Conclusion & Perspectives

Experimental Validation

Experimental Setup

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



↪ Managed to **adapt to** $\neq$ **I/O loads** & main contrib is from model ($\hat{F}$)

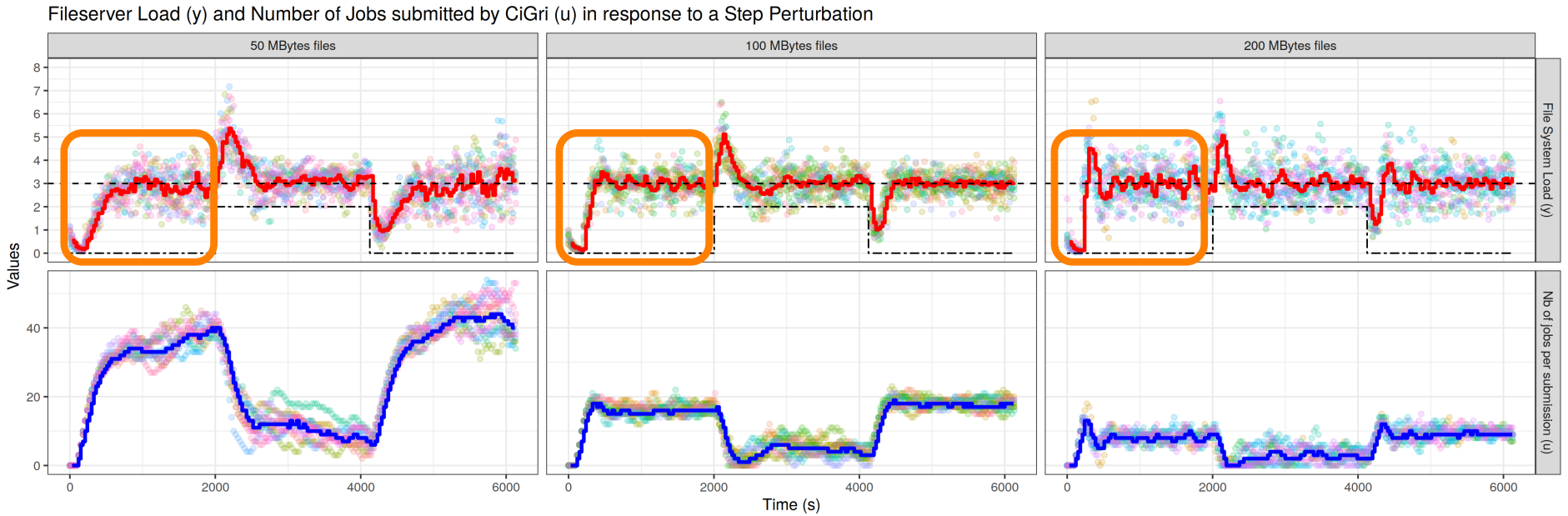
Experimental Validation with Disturbances

Synthetic Load

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

Observations

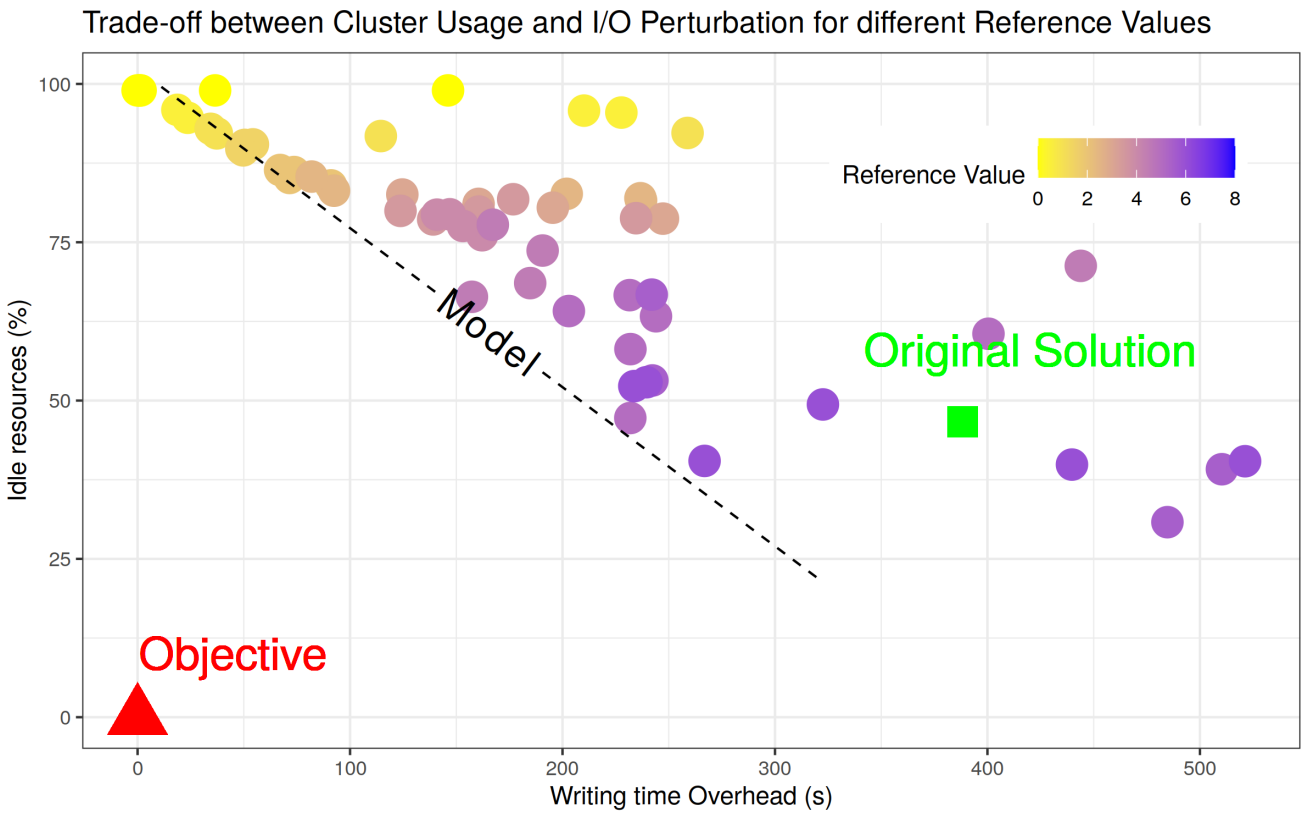
- Good response & tracking
- **but** $\neq$ behaviors for $\neq$ I/O loads
- *e.g.*, overshoot



Comparison with the Original *CiGri* Solution

Comparison

- MADBench2: tool for testing I/O $\rightsquigarrow$ represents Priority Users
- Vary Reference value & Observe perturbation + #idle resources



Observations

$\nearrow$ Harvesting

$\Downarrow$

$\nearrow$ Perturbations

QoS $\propto$ Ref Value

Better ctrl compared to Original Solution

1 Introduction & Context

2 The Problem to Tackle

3 Experimental Validation

4 Conclusion & Perspectives

Conclusion & Perspectives

Reminder of the Objective

Collect **max idle resources** with **min perturbations**

Results

- Used a **Model-Free** approach for dynamical resources harvesting
- Good control and tracking for range of I/O loads
- Trade-off between the harvesting and the perturbations

Perspectives

- Investigate changes of behavior based on I/O loads
- Adapt Reference Value based on #Priority Users