Towards Reusability of Autonomic Controllers in High Performance Computing

GDR GPL - YODA, Vannes

Quentin GUILLOTEAU,[*] Éric RUTTEN,[*] Bogdan ROBU,[**] Olivier RICHARD[*]

[*]Université Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
[**]Université Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab

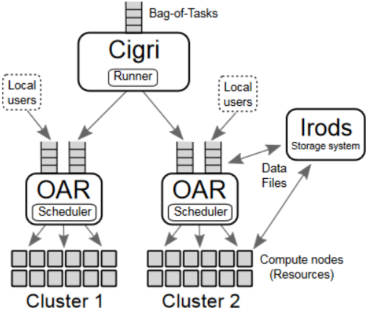
2022-06-08

Context

Idle HPC Resources $\Longrightarrow$ Lost Computing Power $\rightsquigarrow$ **<u>How to Harvest</u> ?**

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

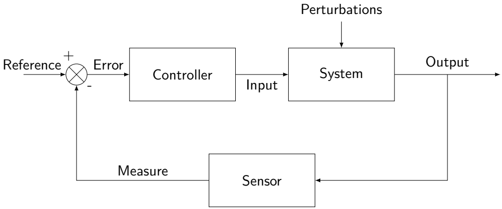
Auto-regulating Systems given **high-level objectives**

Phases: **M**onitor ⇝ **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



1 Introduction & Context

2 The *CiGri* Middleware

3 Design of Controllers
- Proportional-Integral
- Model-Free Control
- Ease of Design/Setup

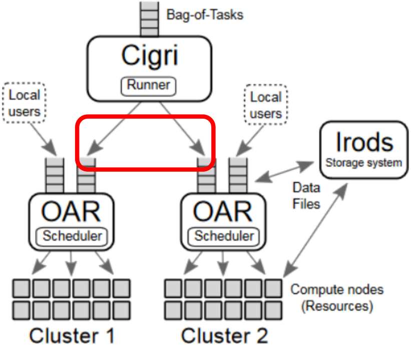
4 Experimental Comparison

5 Conclusion & Perspectives

CiGri: Submission Loop (1/2)

Algorithm 1: Current Solution

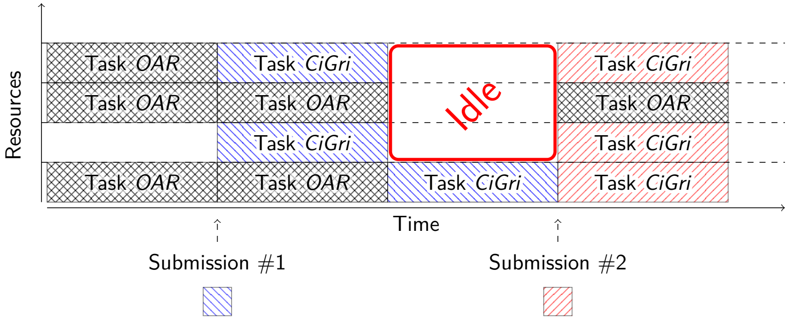
```
$rate = 3$;
$increase\_factor = 1.5$;
while tasks not executed in b-o-t do
    if no task running then
        submit rate tasks;
        $rate = \min(rate \times increase\_factor, 100)$;
    end
    while nb of tasks running > 0 do
        sleep during 30 sec;
    end
end
```



CiGri: Submission (2/2)

The Issue

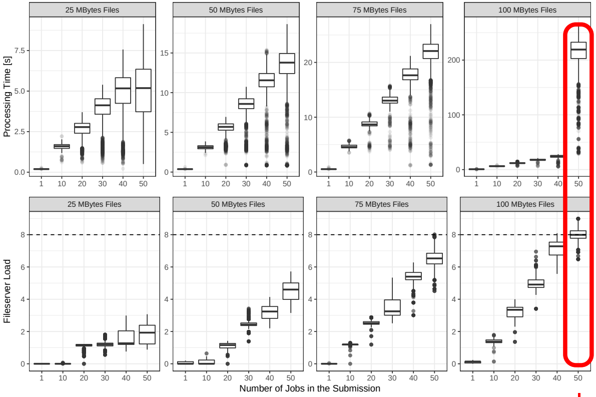
Must wait for termination of the previous submission to submit again
↪ reduce overload but introduce **underutilization** of the resources



Degradation of the File System Performances

↗ Jobs ⟹ ↗ I/O ⟹ ↗ More delay for users ⇝ **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

Our Global Problem and Objectives

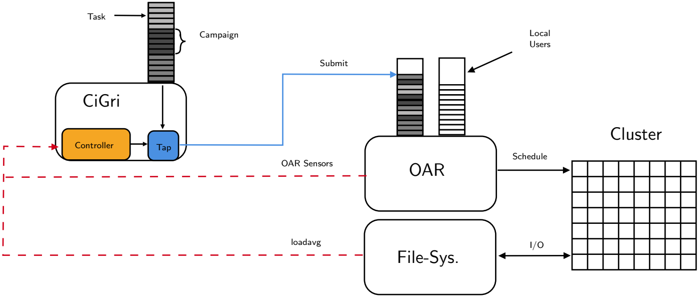
Objective

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

- max cluster utilization
- min perturbations

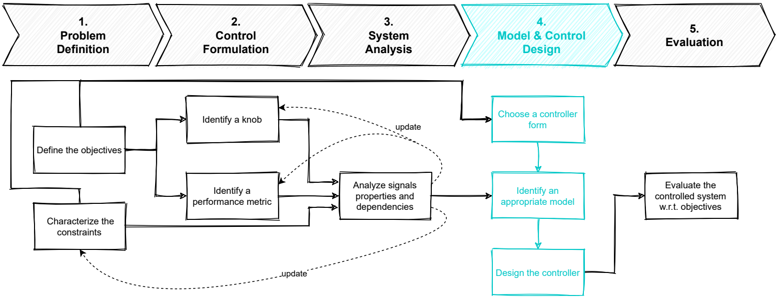
Means

- 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**


1. Problem Definition
2. Control Formulation
3. System Analysis
4. Model & Control Design
5. Evaluation
Define the objectives
Characterize the constraints
Identify a knob
Identify a performance metric
Analyze signals properties and dependencies
update
update
Choose a controller form
Identify an appropriate model
Design the controller
Evaluate the controlled system w.r.t. objectives

However...

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

↪ **Design** Cost? **Setup** Cost? Runtime **Performances**?

Comparison Framework

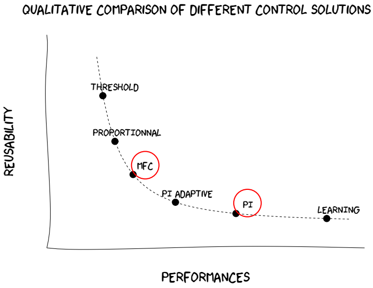
Two Controllers

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

Variations: jobs (I/O, duration), cluster.

Reusability Criterion

- Design Time Cost
- Runtime Performances



Goal

Compare Controllers Reusability: Design Cost vs. Performances

1 Introduction & Context

2 The *CiGri* Middleware

3 Design of Controllers
- Proportional-Integral
- Model-Free Control
- Ease of Design/Setup

4 Experimental Comparison

5 Conclusion & Perspectives

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

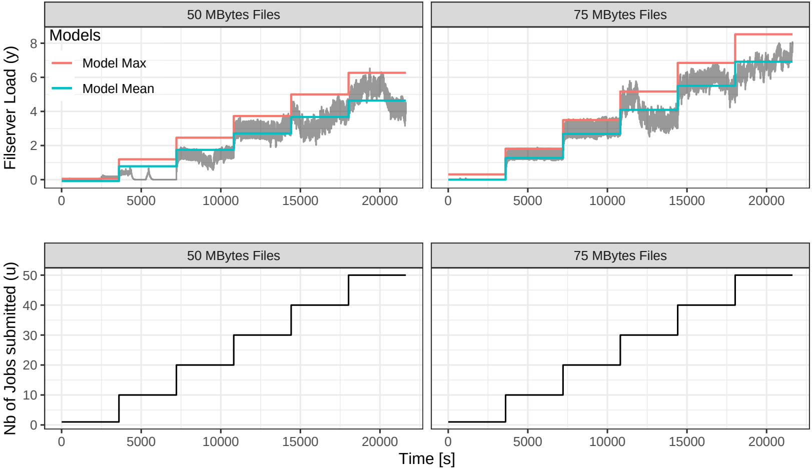
- Actuator: #jobs to sub $\rightsquigarrow$ $\mathbf{u}$
- Sensor: FS Load $\rightsquigarrow$ $\mathbf{y}$
- Error: $Reference - Sensor$

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

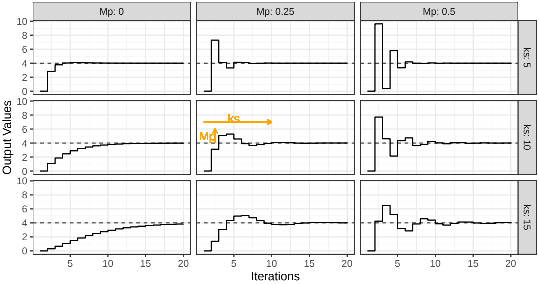


$$y = \alpha + \beta_1 f + \beta_2 u + \gamma f u$$

PI: Closed-Loop Behavior

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

Closed loop behaviour of our system for different values of (ks, Mp)



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 The *CiGri* Middleware

3 Design of Controllers

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

4 Experimental Comparison

5 Conclusion & Perspectives

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

Model-Free Control

- Introduces *intelligent* Controllers (*iPID*)
- Easier to tune than PI
- Adapt to the plant/system
- 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
- α: **non-physical cst parameter**
- $\mathbf{K}_p$: **Gain of the controller**

Choice of Parameters

Empirical choice of parameters (α & $\mathbf{K}_p$)

1 Introduction & Context

2 The *CiGri* Middleware

3 Design of Controllers

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

4 Experimental Comparison

5 Conclusion & Perspectives

Ease of Design/Setup

Proportional-Integral

- Cumbersome to set up
- Requires identification
- Only for identified system

+ Behavior guarantee

Model-Free Control

+ Easy to set up
+ No identification
+ Should adapt to the plant

- No behavior guarantee

Take away

↪ MFC has lighter setup phase, but PI has more guarantees

1 Introduction & Context

2 The *CiGri* Middleware

3 Design of Controllers

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

4 Experimental Comparison

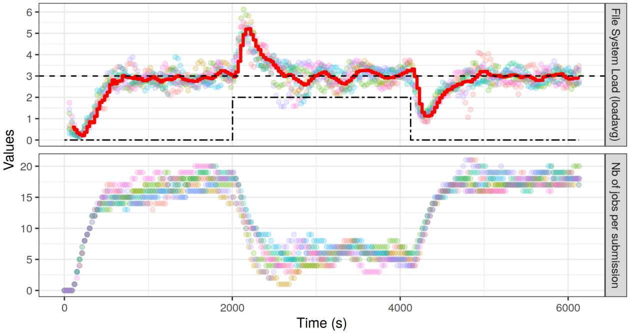
5 Conclusion & Perspectives

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

Fileserver Load and Submission size

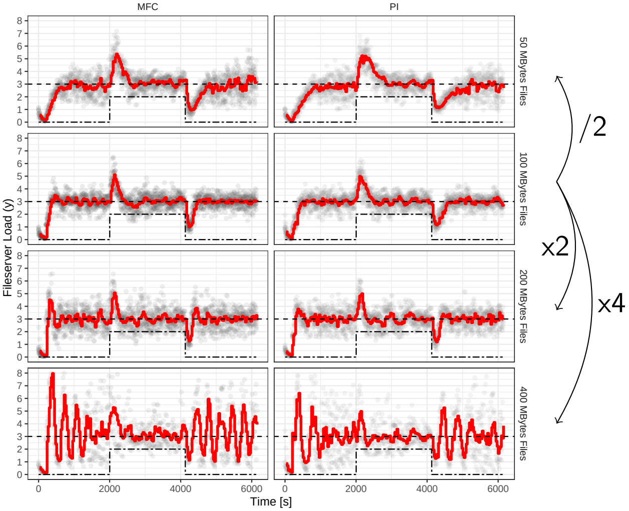


Synthetic Load

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

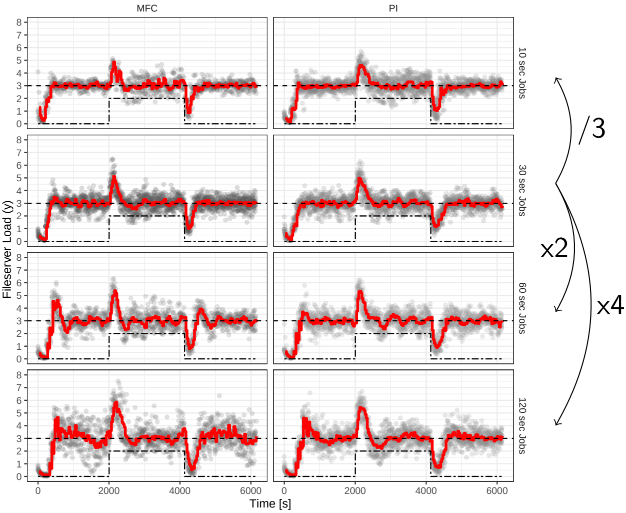
Variation in I/O

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



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

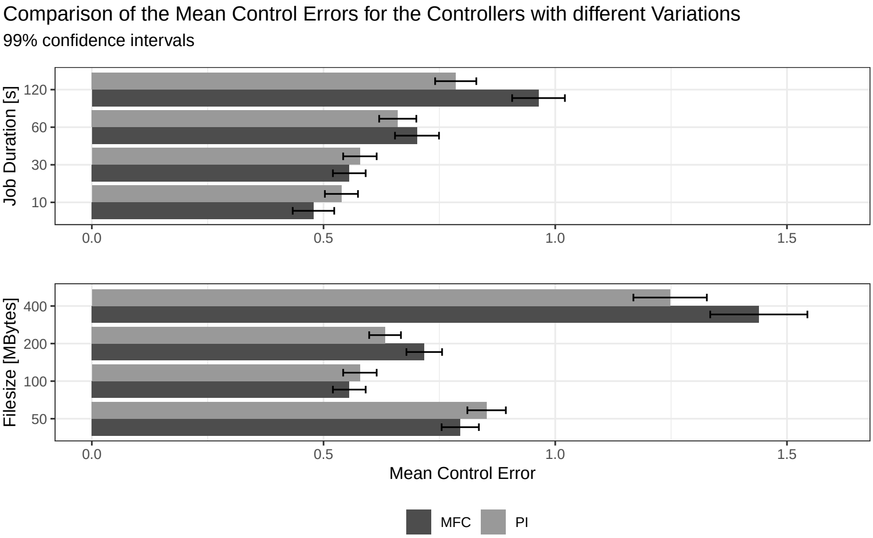
Variation in Execution Time



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

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

Performances Comparison



1 Introduction & Context

2 The *CiGri* Middleware

3 Design of Controllers
- Proportional-Integral
- Model-Free Control
- Ease of Design/Setup

4 Experimental Comparison

5 Conclusion & Perspectives

Conclusion & Perspectives

Reminder of the Objective

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

Results

Compared 2 Controllers: PI & MFC on I/O and job dur. Variations

- MFC has smaller design cost
- $\simeq$ performances for both controllers

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

Perspectives

- Compare with other Solutions (e.g., PI Adaptive, MPC + GP)
- Investigate more variations dimensions (e.g., FS)