Control-based runtime management of HPC systems with support for reproducible experiments

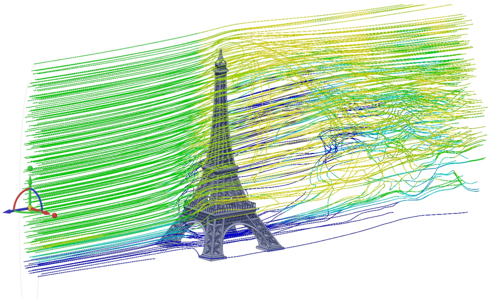
PhD Thesis Defense

Quentin GUILLOTEAU
Ctrl-A and DataMove teams

2023-12-11

Univ. Grenoble Alpes, INRIA, CNRS, LIG
Quentin.Guilloteau@univ-grenoble-alpes.fr

Reviewers: Alexandru COSTAN
Alessandro PAPADOPOULOS
Examiners: Fabienne BOYER
Georges DA COSTA
Noël DE PALMA
Supervisors: Eric RUTTEN
Olivier RICHARD



Computations too demanding $\rightsquigarrow$ need **several powerful** machines
$\hookrightarrow$ expensive $\rightsquigarrow$ shared $\rightsquigarrow$ **reservation process**

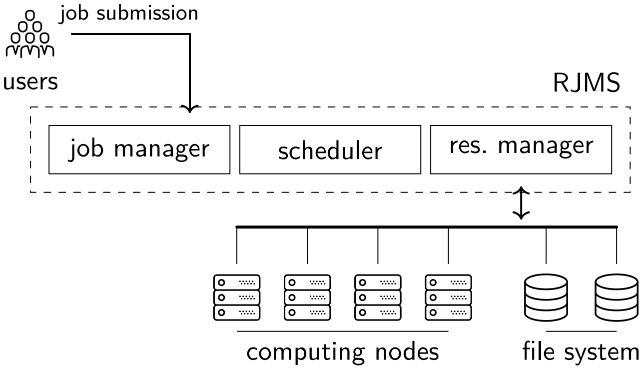
Resouces and Job Management System

HPC Jobs

- Some computations
- Static resource allocation
- Static time allocation

HPC Cluster

- Computing nodes
- Interconnected
- High speed network, I/O



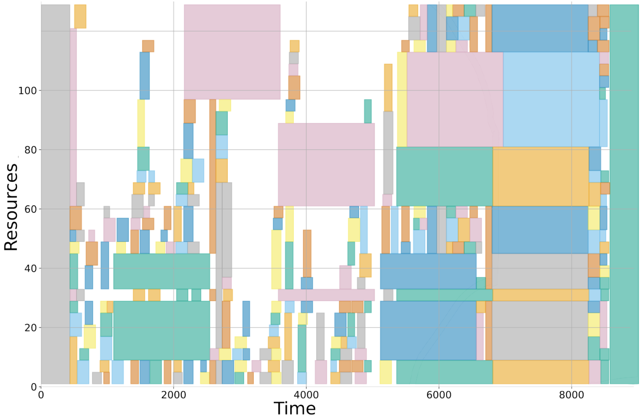
Resources and **J**obs **M**anagement **S**ystem [Ble17]

HPC Jobs

- Some computations
- Static resource allocation
- Static time allocation

HPC Cluster

- Computing nodes
- Interconnected
- High speed network, I/O



Gantt Chart

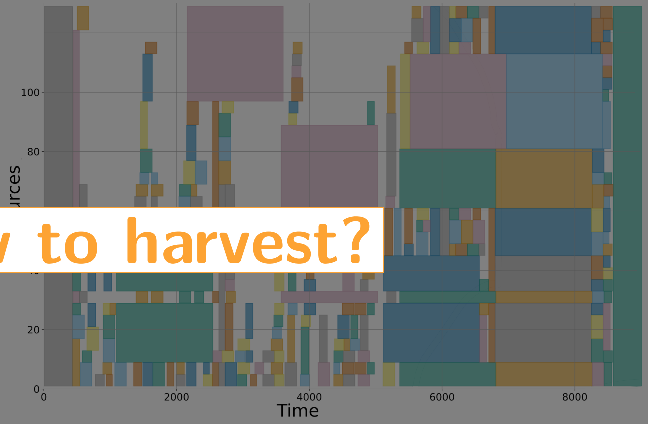
Idle Resources = Wasted Computing Power and Money

HPC Jobs

- Some computations
- Static resource allocation
- Static time allocation

HPC Cluster

- Computing nodes
- Interconnected
- High speed network, I/O



Gantt Chart

$\hookrightarrow$ **How to harvest?**

Idle Resources = Wasted Computing Power and Money

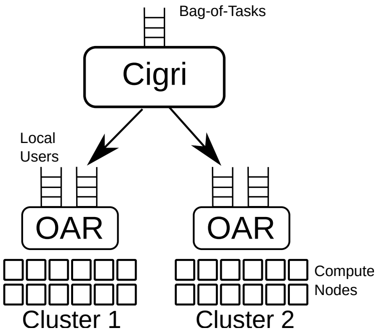
Harvesting Idle Resources

Main idea: Use smaller, killable jobs (e.g., Big Data [Mer+17], FaaS [Prz+22])

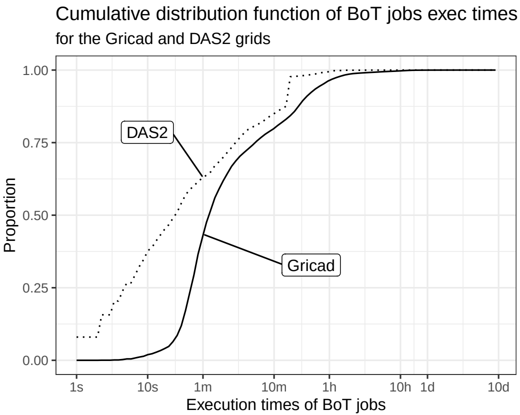
CiGri [GRC07]

- Grid middleware used at Gricad
- **Bag-of-tasks**: many, multi-parametric
- **Best-effort Jobs**: Lowest priority
- **Objectives**:
  - Collect grid idle resources
  - Reduce pressure on RJMS
- Submits like a *periodic tap*
  - submits jobs then,
  - waits for *all* jobs to terminate

  ↪ **suboptimal**!



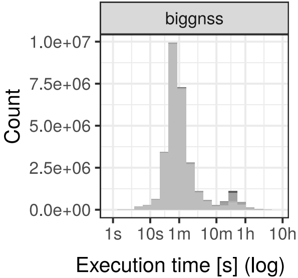
CiGri jobs [GRR22]



10 years, 44 Millions jobs

Example: BigGNSS [Dép+18]

- A lot of satellites $\Longrightarrow$ a lot of data
- Several stations $\rightsquigarrow$ **Campaigns**
- Subdivision of the processing $\rightsquigarrow$ **Jobs**
- Unique binary + different inputs



Problem

$\nearrow$ Harvesting $\Longrightarrow$ $\nearrow$ Performance Degradation $\rightsquigarrow$ **Trade-off**

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

Problem

$\nearrow$ Harvesting $\implies$ $\nearrow$ Performance Degradation $\rightsquigarrow$ **Trade-off**

$\hookrightarrow$ Unpredictability $\implies$ **runtime management**

In this PhD thesis

1. How to **submit** CiGri jobs to harvest idle resources with **controlled** degradation for priority users?

Problem

$\nearrow$ Harvesting $\implies$ $\nearrow$ Performance Degradation $\rightsquigarrow$ **Trade-off**

$\hookrightarrow$ Unpredictability $\implies$ **runtime management**

In this PhD thesis

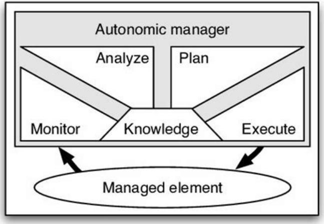
1. How to **submit** CiGri jobs to harvest idle resources with **controlled** degradation for priority users?
2. How to improve the **cost** and **reproducibility** of experiments on grid/cluster systems?

Harvesting idle resources

Runtime Management: Autonomic Computing (AC)

AC and the MAPE-K Loop [KC03]

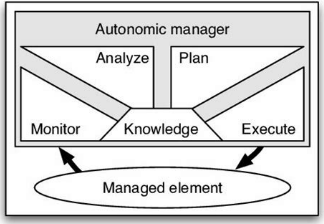
- **Auto-regulation** given **high-level objectives**
- implementations: rules, AI, etc.



Runtime Management: Autonomic Computing (AC) and Control Theory

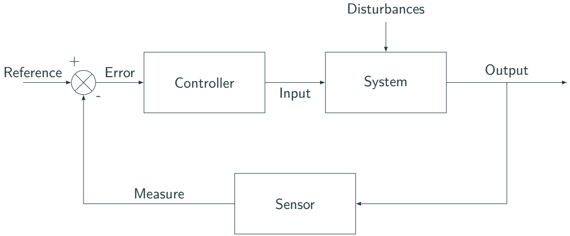
AC and the MAPE-K Loop [KC03]

- **Auto-regulation** given **high-level objectives**
- implementations: rules, AI, etc.



Control Theory

- Regulate dynamical systems
- physical systems
- mathematically proven properties
- performance, robustness, **explainability**

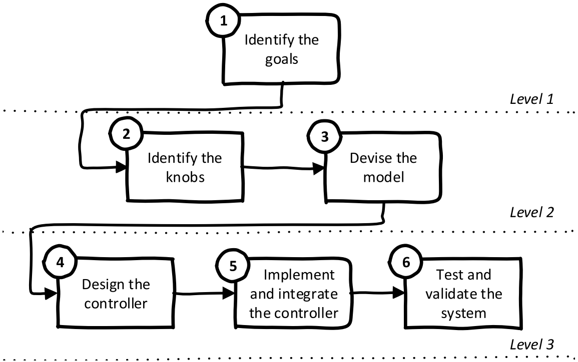


1. Identify the goals

Problem formulation

- Use Control Theory to....
- ...harvest idle resources...
- ...in a **non-intrusive** way
- max cluster utilization
- min degradation of performance

↪ Focus on I/O degradation



Steps to design a controller [Fil+15]

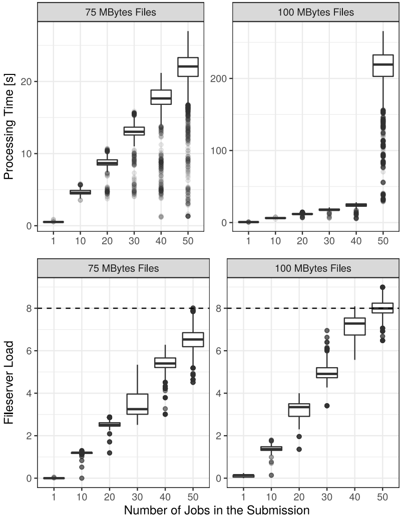
2. Identify the knobs

Actuators (u)

Number of jobs submitted by CiGri

Sensors (y)

- File-System (NFS):
  - **indirect** measure of overhead
  - `/proc/loadavg` [FZ87]
    - $\simeq$ number of processes running
    - well known by system administrators
    - Exponential Smoothing $\rightsquigarrow$ Inertia
      $\hookrightarrow$ Nice for the control



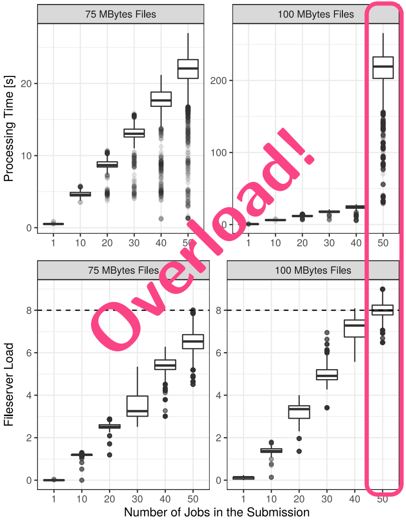
2. Identify the knobs

Actuators (u)

Number of jobs submitted by CiGri

Sensors (y)

- File-System (NFS):
  - **indirect** measure of overhead
  - `/proc/loadavg` [FZ87]
    - ≃ number of processes running
    - well known by system administrators
    - Exponential Smoothing ⇝ Inertia
      ↪ Nice for the control
  - **know limits of sensor**



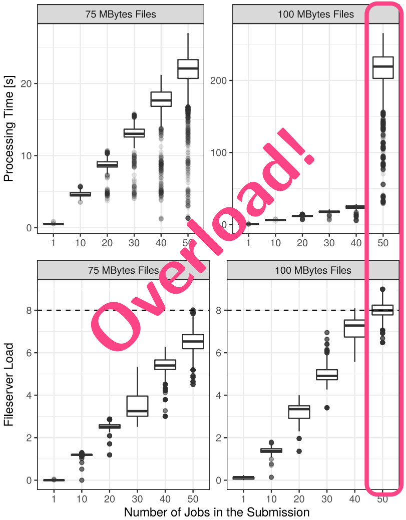
2. Identify the knobs

Actuators (u)

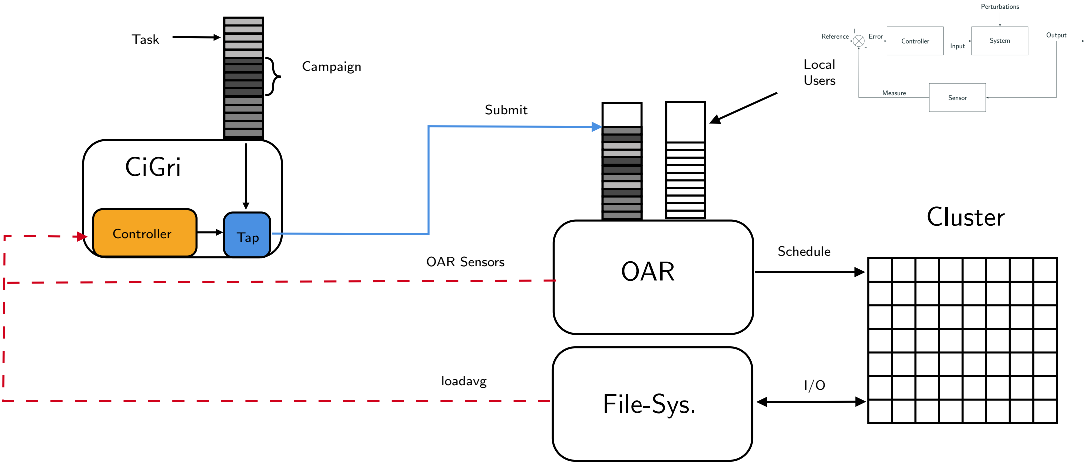
Number of jobs submitted by CiGri

Sensors (y)

- File-System (NFS):
  - **indirect** measure of overhead
  - /proc/loadavg [FZ87]
    - ≃ number of processes running
    - well known by system administrators
    - Exponential Smoothing ⇝ Inertia
      ↪ Nice for the control
  - **know limits of sensor**
- Cluster: OAR API (nb running, waiting jobs)



Feedback loop in CiGri



Reference value: acceptable `load` on the File-System, chosen by system admins

3. Devise the model

First, a Model ... (i.e., how does the system behave without Control)

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

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

$$\mathbf{u}(k) = \mathbf{K}_p \times Error(k)$$

Sensors & Actuators

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

Methodology

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

3. Devise the model

First, a Model ... (i.e., how does the system behave without Control)

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

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

$$\mathbf{u}(k) = \mathbf{K}_p \times Error(k) + \mathbf{K}_i \times \sum_{i}^{k} Error(i)$$

Sensors & Actuators

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

Methodology

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

3. Devise the model

First, a Model ... (i.e., how does the system behave without Control)

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

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

$$\mathbf{u}(k) = \mathbf{K}_p \times Error(k) + \mathbf{K}_i \times \sum_{i}^{k} Error(i) + \mathbf{K}_d \times (Error(k) - Error(k-1))$$

Sensors & Actuators

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

Methodology

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

3. Devise the model

First, a Model ... (i.e., how does the system behave without Control)

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

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

$$\mathbf{u}(k) = \mathbf{K}_p \times Error(k) + \mathbf{K}_i \times \sum_{i}^{k} Error(i) + \mathbf{K}_d \times (Error(k) - Error(k-1))$$

Sensors & Actuators

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

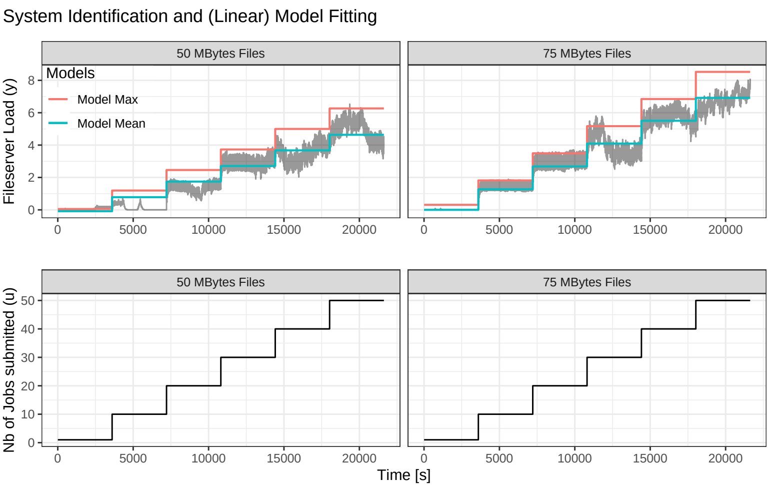
Methodology

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

3. Devise the model - Open-Loop Experiments

- "step" inputs
- $\neq$ I/O loads (f)
- observe behavior
- **linear** model

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



3. Devise the model - First order model

First order model: $\mathbf{y}(k+1) = a \times \mathbf{y}(k) + b \times \mathbf{u}(k) \rightsquigarrow a, b = ?$

3. Devise the model - First order model

First order model: $\mathbf{y}(k+1) = a \times \mathbf{y}(k) + b \times \mathbf{u}(k) \rightsquigarrow a, \mathbf{b} = ?$ (a from def of `loadavg`)

3. Devise the model - First order model

First order model: $\mathbf{y}(k+1) = a \times \mathbf{y}(k) + b \times \mathbf{u}(k) \rightsquigarrow a, \mathbf{b} = ?$ (a from def of `loadavg`)

In steady state (ss)

$$\mathbf{y}_{ss} = a \times \mathbf{y}_{ss} + b \times \mathbf{u}_{ss}$$

3. Devise the model - First order model

First order model: $\mathbf{y}(k+1) = a \times \mathbf{y}(k) + b \times \mathbf{u}(k) \rightsquigarrow a, \mathbf{b} = ?$ (a from def of `loadavg`)

In steady state (ss)

$$\mathbf{y}_{ss} = a \times \mathbf{y}_{ss} + b \times \mathbf{u}_{ss} \implies b = \frac{\mathbf{y}_{ss} \times (1 - a)}{\mathbf{u}_{ss}}$$

3. Devise the model - First order model

First order model: $\mathbf{y}(k+1) = a \times \mathbf{y}(k) + b \times \mathbf{u}(k) \rightsquigarrow a, \mathbf{b} = ?$ (a from def of `loadavg`)

In steady state (ss)

$$\mathbf{y}_{ss} = a \times \mathbf{y}_{ss} + b \times \mathbf{u}_{ss} \implies b = \frac{\mathbf{y}_{ss} \times (1-a)}{\mathbf{u}_{ss}} \implies \frac{(\alpha + \beta_1 f + \beta_2 \mathbf{u}_{ss} + \gamma f \mathbf{u}_{ss}) \times (1-a)}{\mathbf{u}_{ss}}$$

First order model: $\mathbf{y}(k+1) = a \times \mathbf{y}(k) + b \times \mathbf{u}(k) \leadsto a, \mathbf{b} = ?$ (a from def of `loadavg`)

In steady state (ss)

$$\mathbf{y}_{ss} = a \times \mathbf{y}_{ss} + b \times \mathbf{u}_{ss} \implies b = \frac{\mathbf{y}_{ss} \times (1-a)}{\mathbf{u}_{ss}} \implies \frac{(\alpha + \beta_1 f + \beta_2 \mathbf{u}_{ss} + \gamma f \mathbf{u}_{ss}) \times (1-a)}{\mathbf{u}_{ss}}$$

$$\implies b \simeq (\beta_2 + \gamma f) \times (1-a)$$

3. Devise the model - First order model

First order model: $\mathbf{y}(k+1) = a \times \mathbf{y}(k) + b \times \mathbf{u}(k) \rightsquigarrow a, \mathbf{b} = ?$ (a from def of `loadavg`)

In steady state (ss)

$$\mathbf{y}_{ss} = a \times \mathbf{y}_{ss} + b \times \mathbf{u}_{ss} \implies b = \frac{\mathbf{y}_{ss} \times (1-a)}{\mathbf{u}_{ss}} \implies \frac{(\alpha + \beta_1 f + \beta_2 \mathbf{u}_{ss} + \gamma f \mathbf{u}_{ss}) \times (1-a)}{\mathbf{u}_{ss}}$$

$$\implies b \simeq (\beta_2 + \gamma f) \times (1-a)$$

Where are we?

Open-Loop Experiments ✓

First order model: $\mathbf{y}(k+1) = a \times \mathbf{y}(k) + b \times \mathbf{u}(k) \leadsto a, \mathbf{b} = ?$ (a from def of `loadavg`)

In steady state (ss)

$$\mathbf{y}_{ss} = a \times \mathbf{y}_{ss} + b \times \mathbf{u}_{ss} \implies b = \frac{\mathbf{y}_{ss} \times (1-a)}{\mathbf{u}_{ss}} \implies \frac{(\alpha + \beta_1 f + \beta_2 \mathbf{u}_{ss} + \gamma f \mathbf{u}_{ss}) \times (1-a)}{\mathbf{u}_{ss}}$$

$$\implies b \simeq (\beta_2 + \gamma f) \times (1-a)$$

Where are we?

Open-Loop Experiments ✓ $\leadsto$ Model (1st order) $\mathbf{y}(k+1) = a \times \mathbf{y}(k) + b \times \mathbf{u}(k)$ ✓

3. Devise the model - First order model

First order model: $\mathbf{y}(k+1) = a \times \mathbf{y}(k) + b \times \mathbf{u}(k) \leadsto a, \mathbf{b} = ?$ (a from def of `loadavg`)

In steady state (ss)

$$\mathbf{y}_{ss} = a \times \mathbf{y}_{ss} + b \times \mathbf{u}_{ss} \implies b = \frac{\mathbf{y}_{ss} \times (1-a)}{\mathbf{u}_{ss}} \implies \frac{(\alpha + \beta_1 f + \beta_2 \mathbf{u}_{ss} + \gamma f \mathbf{u}_{ss}) \times (1-a)}{\mathbf{u}_{ss}}$$

$$\implies b \simeq (\beta_2 + \gamma f) \times (1-a)$$

Where are we?

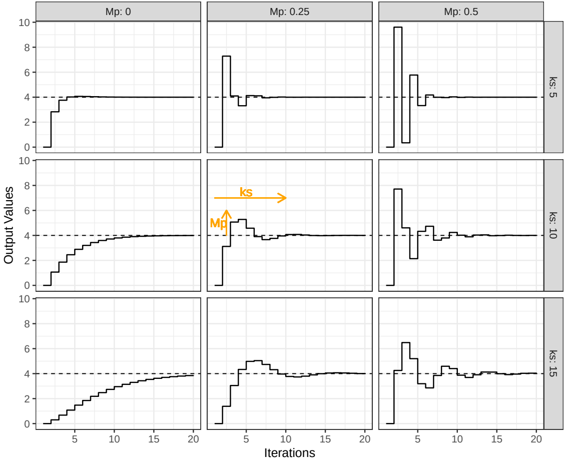
Open-Loop Experiments ✓ $\leadsto$ Model (1st order) $\mathbf{y}(k+1) = a \times \mathbf{y}(k) + b \times \mathbf{u}(k)$ ✓ $\leadsto$ Controller Gains $\mathbf{K}_p, \mathbf{K}_i, \mathbf{K}_d$

4. Design the controller

Controller Gains are ...
functions of the model and

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

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

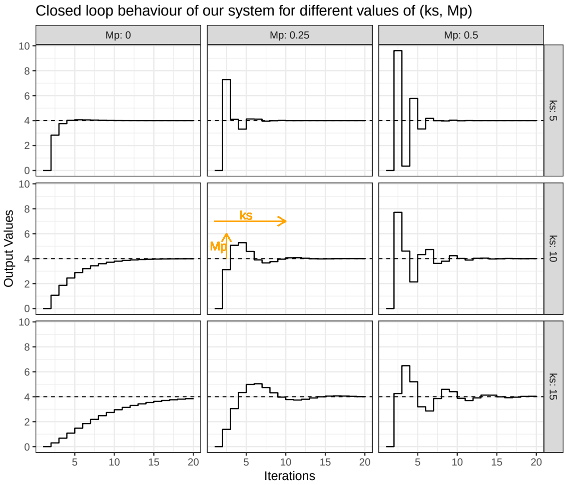


4. Design the controller

Controller Gains are ...
functions of the model and

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

Can choose the behavior!



4. Design the controller

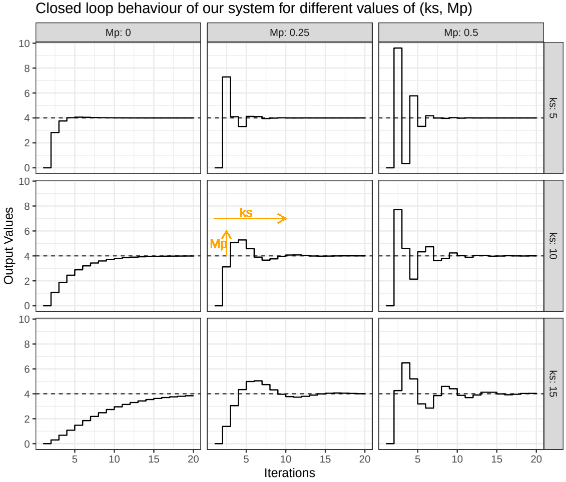
Controller Gains are ...
functions of the model and

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

Can choose the behavior!

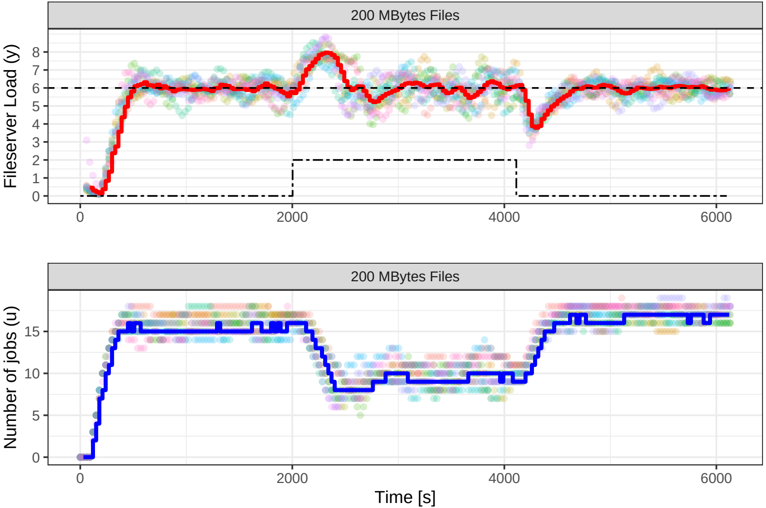
Non-Intrusive Harvesting

- no overshoot
- but "fast" response



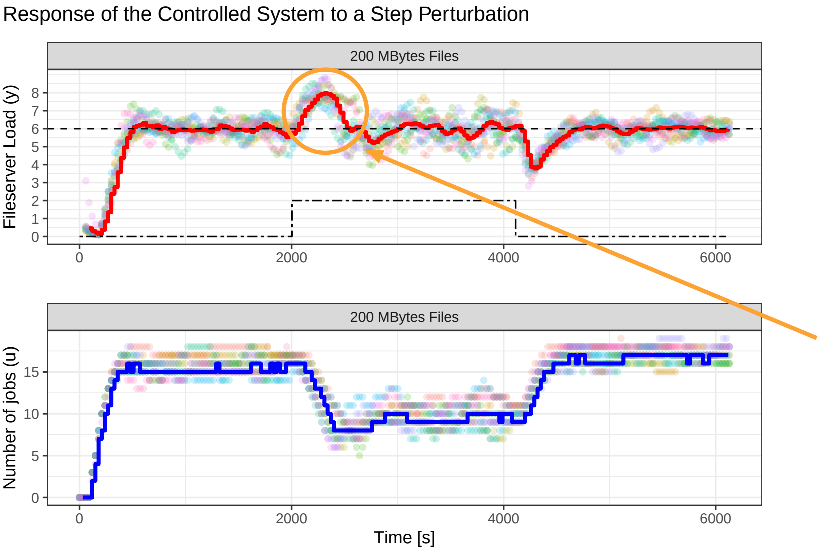
5./6. Implement and validate the controller - Evaluation with synthetic jobs

Response of the Controlled System to a Step Perturbation



- constant reference
- synthetic jobs
- step disturbance

Manage to control the load of the File-System



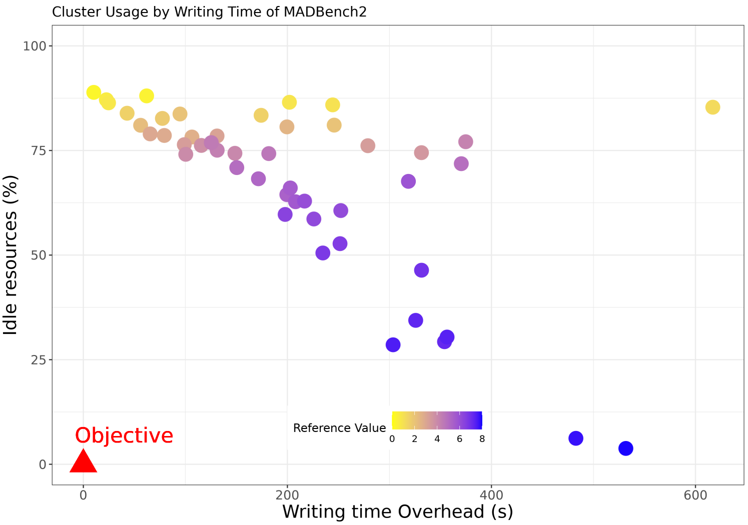
- constant reference
- synthetic jobs
- step disturbance

Manage to control the load of the File-System

takes time to react

↪ might cause **overload**

Trade-off: Idleness versus Performance degradation (I/O Overhead)



- MADBench2 [Bor+07]
- various reference values
- compute idle resources
- compute I/O overhead

Trade-off between Harvesting & I/O overhead through the reference value

A note on controllers' reusability

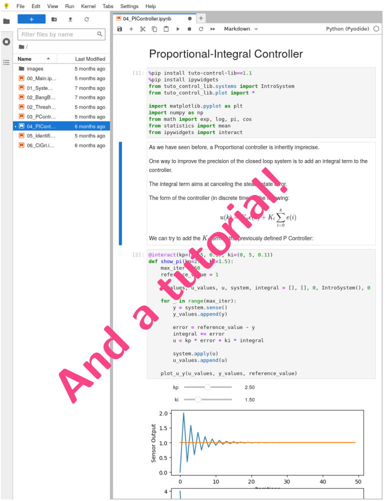
- Controllers **linked** to the identified system
- what if new cluster? new configuration?
- Grid/Cluster administrators
  ↪ **not** control theory experts!
- compared 3 controllers (w.r.t. portability, guarantees, competence required)
- example: Portability vs. Performance

↪ **gave recommendations** for system administrators

A note on controllers’ reusability

- Controllers **linked** to the identified system
- what if new cluster? new configuration?
- Grid/Cluster administrators
  ↪ **not** control theory experts!
- compared 3 controllers (w.r.t. portability, guarantees, competence required)
- example: Portability vs. Performance

↪ **gave recommendations** for system administrators



Control-based harvesting of idle resources: Wrapping up

Objectives

- Control CiGri submissions based on File-System load ✓
- Control CiGri submissions to reduce idle/killed wasted time ✓
- Can merge controllers! (with some subtelties)
- Guidelines for system administrators ✓
- Tutorial to introduce control theory to computer scientists ✓

Limitations and Perspectives

- Tested with *synthetic* jobs $\rightsquigarrow$ real trace
- Need more info about CiGri jobs' I/O patterns
- Submissions to several clusters
- Sensor for Parallel File-System (PFS) ?

Objectives

- Control CiGri submissions based on File-System load ✓
- Control CiGri submissions to reduce idle/killed wasted time ✓
- Can merge controllers! (with some subtelties)
- Guidelines for system administrators ✓

Take Away: Control Theory **valuable approach** to exploit such trade-offs

- Tutorial to introduce control theory to computer scientists ✓

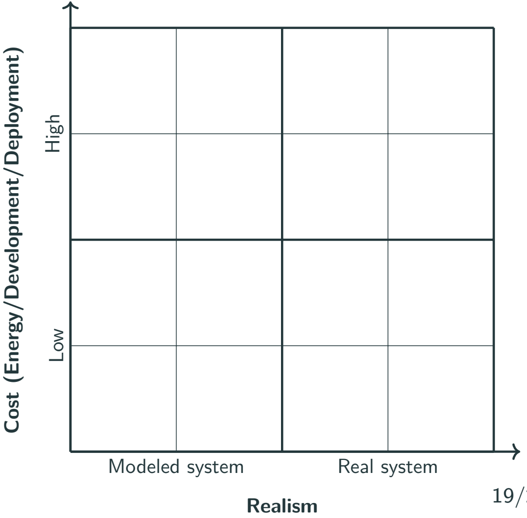
Limitations and Perspectives

- Tested with *synthetic* jobs $\rightsquigarrow$ real trace
- Need more info about CiGri jobs' I/O patterns
- Submissions to several clusters
- Sensor for Parallel File-System (PFS) ?

Experiment costs and reproducibility

A grid middleware needs . . . a grid!

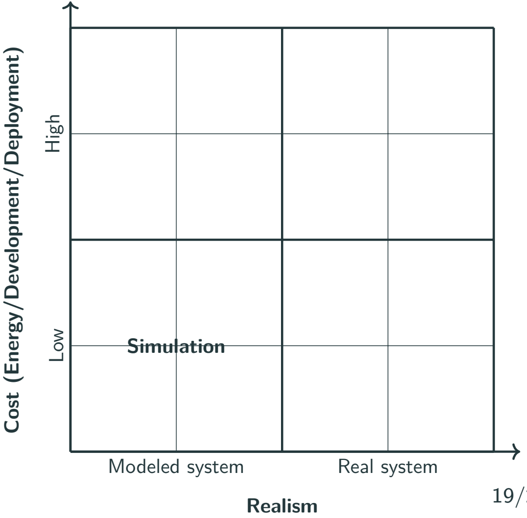
How many machines required to perform **realistic** experiments on a grid middleware like CiGri?



A grid middleware needs . . . a grid!

How many machines required to perform **realistic** experiments on a grid middleware like CiGri?

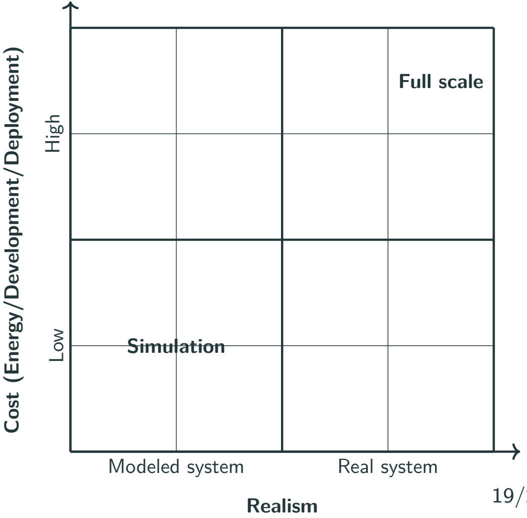
- **Simulation**: fast ☺, modeled ☹, poor sensor support ☹, poor PFS support ☹



A grid middleware needs . . . a grid!

How many machines required to perform **realistic** experiments on a grid middleware like CiGri?

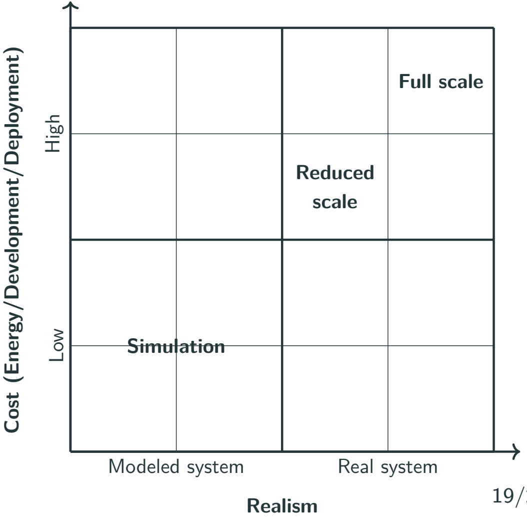
- **Simulation**: fast ☺, modeled ☹, poor sensor support ☹, poor PFS support ☹
- **Full scale**: real environment ☺, expensive and difficult ☹



A grid middleware needs . . . a grid!

How many machines required to perform **realistic** experiments on a grid middleware like CiGri?

- **Simulation**: fast ☺, modeled ☹, poor sensor support ☹, poor PFS support ☹
- **Full scale**: real environment ☺, expensive and difficult ☹
- **Reduced scale**: real environment ☺, cheaper ☺, **realistic ?** ☹

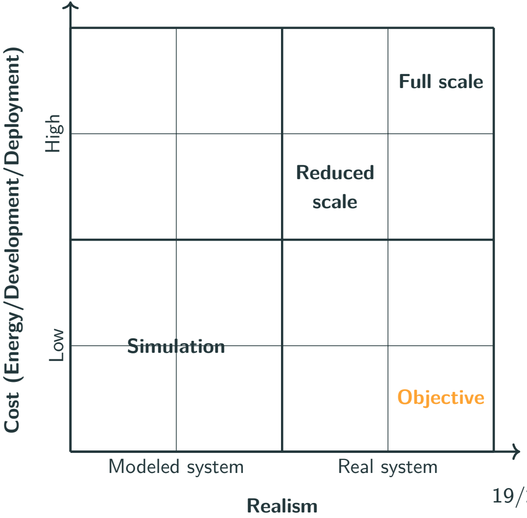


A grid middleware needs . . . a grid!

How many machines required to perform **realistic** experiments on a grid middleware like CiGri?

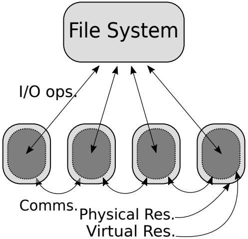
- **Simulation**: fast ☺, modeled ☹, poor sensor support ☹, poor PFS support ☹
- **Full scale**: real environment ☺, expensive and difficult ☹
- **Reduced scale**: real environment ☺, cheaper ☺, **realistic ?** ☹

Objective: Low cost, realist experiments on the real system



Emulating a full scale cluster by folding its deployment [Gui+23]

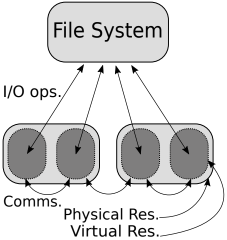
The idea: Deploy more "virtual" resources on one physical machine (≃ oversubscribing)



Scale 1:1

Emulating a full scale cluster by folding its deployment [Gui+23]

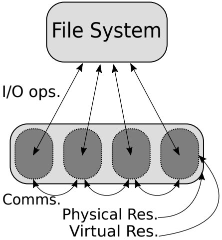
The idea: Deploy more "virtual" resources on one physical machine (≃ oversubscribing)



Scale 1:2

Emulating a full scale cluster by folding its deployment [Gui+23]

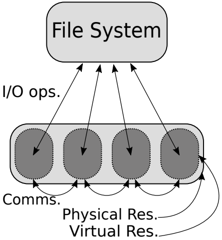
The idea: Deploy more "virtual" resources on one physical machine (≃ oversubscribing)



Scale 1:4

Emulating a full scale cluster by folding its deployment [Gui+23]

The idea: Deploy more "virtual" resources on one physical machine (≃ oversubscribing)

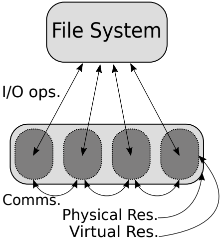


Scale 1:4

+ less resources deployed
+ represents full scale system
+ real system/environment

Emulating a full scale cluster by folding its deployment [Gui+23]

The idea: Deploy more "virtual" resources on one physical machine (≃ oversubscribing)



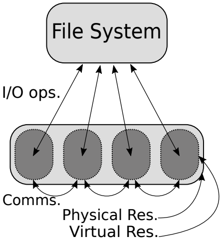
Scale 1:4

+ less resources deployed
+ represents full scale system

+ real system/environment
- **new job model**: `sleep` + `dd`

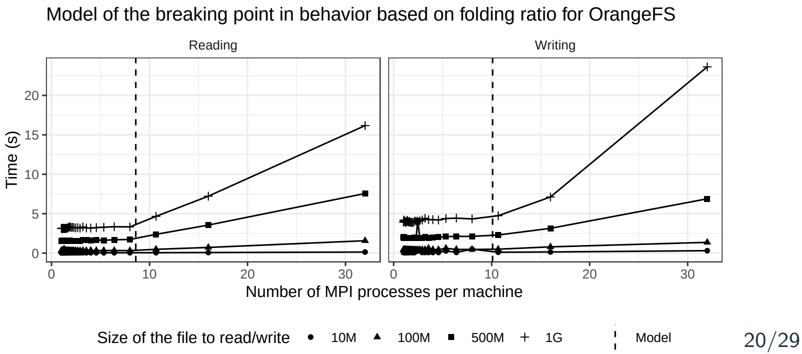
Emulating a full scale cluster by folding its deployment [Gui+23]

The idea: Deploy more "virtual" resources on one physical machine (≃ oversubscribing)



Scale 1:4

+ less resources deployed
+ represents full scale system
+ real system/environment
- **new job model**: `sleep + dd`

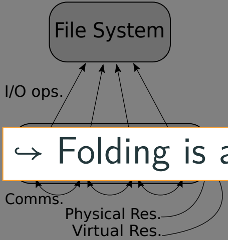


Protocole:

- IOR [Cal23]
- increase folding
- NFS, OrangeFS

Emulating a full scale cluster by folding its deployment [Gui+23]

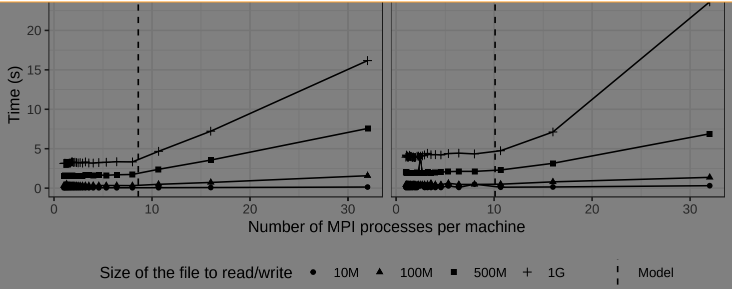
The idea: Deploy more "virtual" resources on one physical machine (≃ oversubscribing)



Scale 1:4

- \+ less resources deployed
- \+ represents full scale system
- \+ real system/environment
- \- **new job model**: `sleep` + `dd`

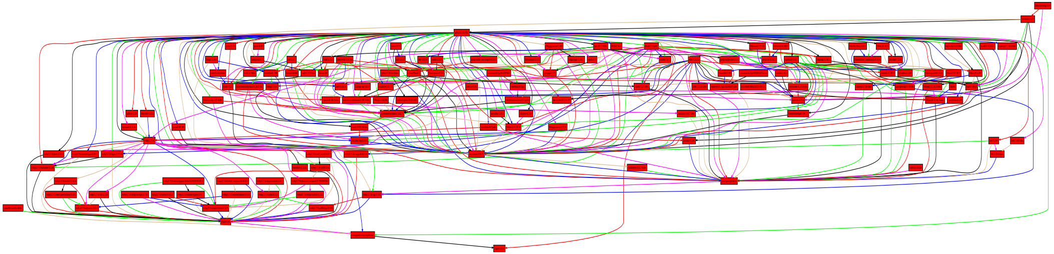
↪ Folding is appropriate until a **breaking point** that we can model



Protocole:

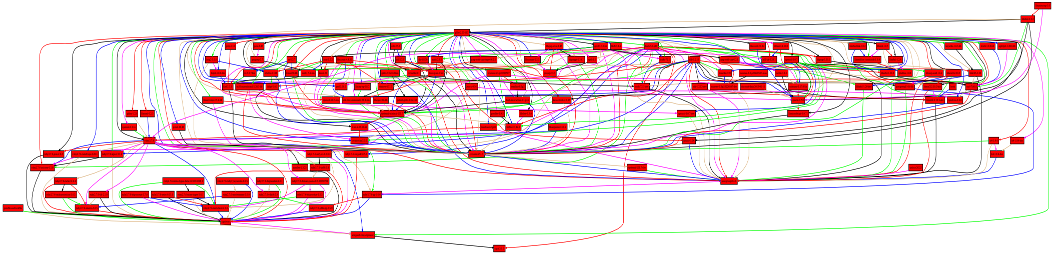
- IOR [Cal23]
- increase folding
- NFS, OrangeFS

Complex Software Environments



Graph of CiGri's software dependencies

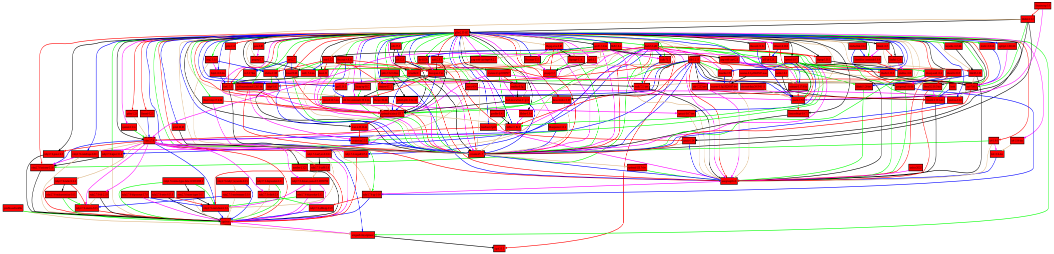
Complex Software Environments



Graph of CiGri's software dependencies

↪ and RJMS, PFS, jobs, etc. ⇝ very complex to **manage/modify**

Complex Software Environments



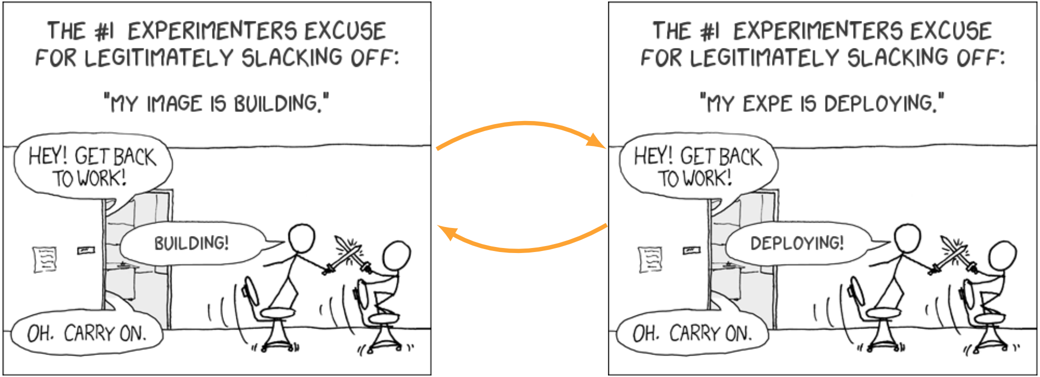
Graph of CiGri's software dependencies

↪ and RJMS, PFS, jobs, etc. ⇝ very complex to **manage/modify**

How to **develop/deploy easily** complex software environments in a **reproducible** fashion?

Generating Distributed Software Environments

↪ **Difficult**, **Time-consuming**, **Script-based** tools, and **Iterative** process

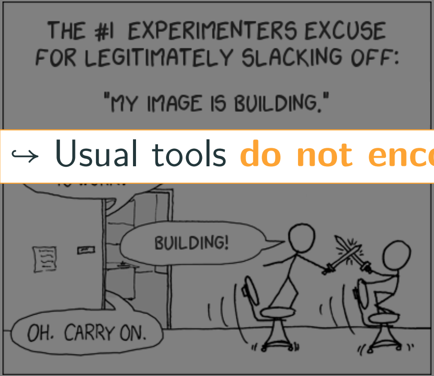


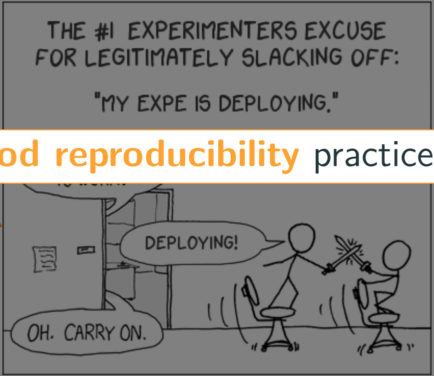
$\simeq$ **10/15 mins** $\qquad$ $\simeq$ **5/10 mins**

↪ Easy to **depend on an external state**: base image, apt mirror, `git` repository

Generating Distributed Software Environments

↪ **Difficult**, **Time-consuming**, **Script-based** tools, and **Iterative** process





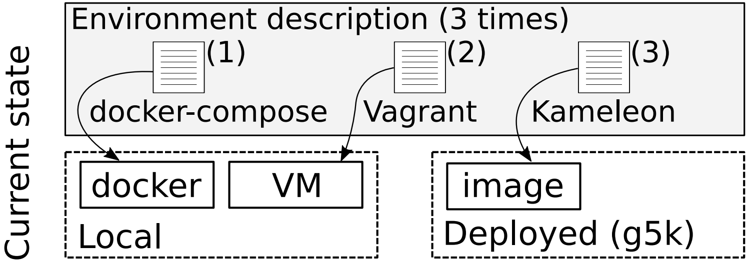
↪ Usual tools **do not encourage good reproducibility** practices

$\simeq$ **10/15 mins**

$\simeq$ **5/10 mins**

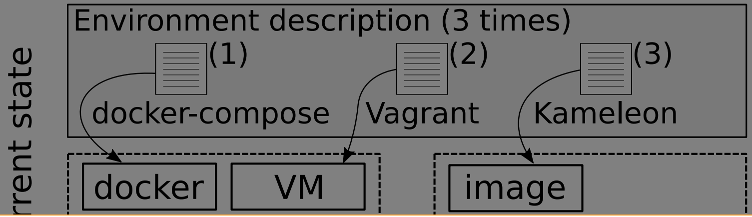
↪ Easy to **depend on an external state**: base image, apt mirror, git repository

One tool, One platform



'So essentially, I want to create a debian12-nfs.qcow2 for VMs equivalent to grid5000's debian12-nfs image. One ***painful way*** *to achieve this would be to install every single thing using the package manager and resolving conflicts by hand.'* *(Grid'5000 User, 2023)*

One tool, One platform

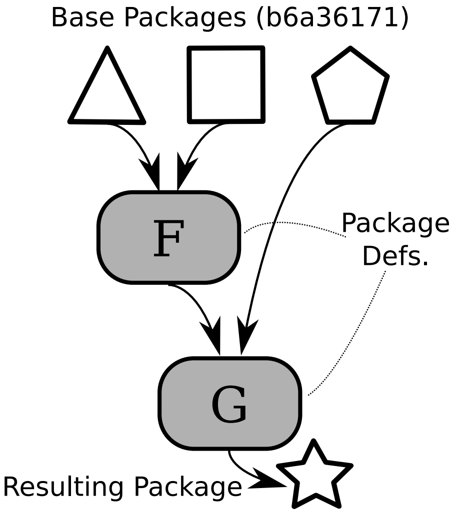


↪ Be able to **develop** distributed environments **locally** and then export

'So essentially, I want to create a debian12-nfs.qcow2 for VMs equivalent to grid5000's debian12-nfs image. One ***painful way*** *to achieve this would be to install every single thing using the package manager and resolving conflicts by hand.'* *(Grid'5000 User, 2023)*

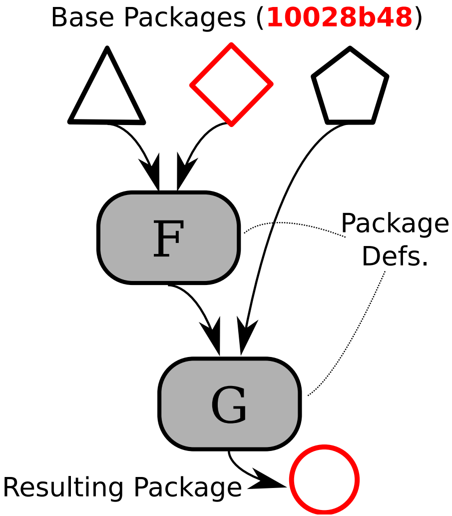
Functional package managers

- Nix, Guix **reproducible by design!**
- packages = functions
  - inputs = dependencies
  - body = commands to build the package
- base packages defined in Git
- sandbox, no side effect
- `/nix/store/`*hash(inputs)*`-my-pkg`
- immutable, read-only
- **precise** definition of `$PATH`
- **can build: container, VM, *system images***



Functional package managers

- Nix, Guix **reproducible by design!**
- packages = functions
  - inputs = dependencies
  - body = commands to build the package
- base packages defined in Git
- sandbox, no side effect
- `/nix/store/`*hash(inputs)*`-my-pkg`
- immutable, read-only
- **precise** definition of `$PATH`
- **can build: container, VM, *system images***



NixOS Compose [Gui+22]

```
{ pkgs, ... }:
let k3sToken = "..."; in {
  roles = {
    server = { pkgs, ... }: {
      environment.systemPackages = with pkgs; [
        k3s gzip
      ];
      networking.firewall.allowedTCPPorts = [
        6443
      ];
      services.k3s = {
        enable = true;
        role = "server";
        package = pkgs.k3s;
        extraFlags = "--agent-token ${k3sToken}";
      };
    };
    agent = { pkgs, ... }: {
      environment.systemPackages = with pkgs; [
        k3s gzip
      ];
      services.k3s = {
        enable = true;
        role = "agent";
        serverAddr = "https://server:6443";
        token = k3sToken;
      };
    };
  };
}
```

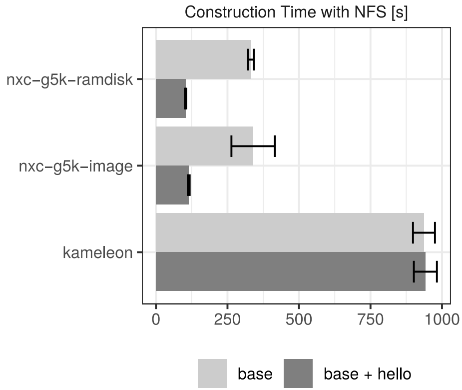
Packages

Ports

Services

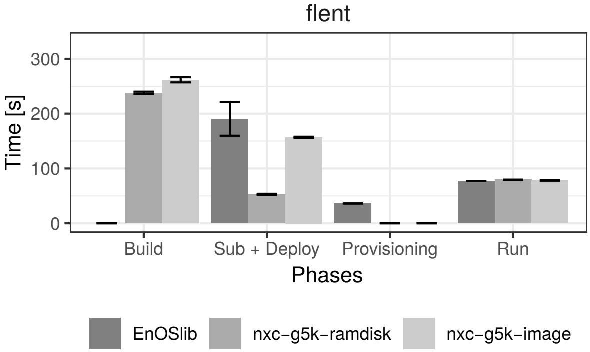
- Python + Nix ($\simeq$ 4000 l.o.c.)
- developing/deploying distributed systems
- **single description** (in Nix), multiple targets
- docker-compose, VM, ramdisk, system image
- can **quickly** setup distributed envs **locally!**
- build, deploy, connect: **unique interface**
- contextualization (ssh keys, `/etc/hosts`, etc.)
- integration with Execo [Imb+13]
- a few, but happy, users ☺

Kameleon [Rui+15]



build ⇝ modify (add `hello`) ⇝ build

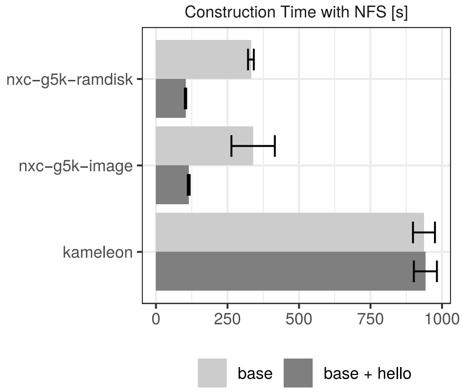
EnOSlib [Che+22]



NixOS Compose ⇝ provisioning done in image

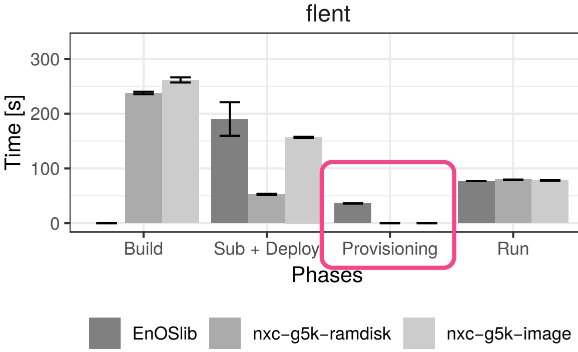
Comparisons - Setting up a distributed environments on Grid'5000

Kameleon [Rui+15]



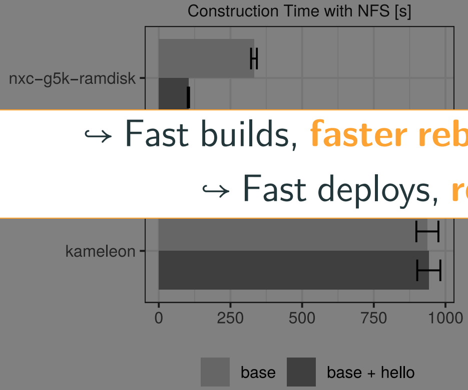
build ⇝ modify (add `hello`) ⇝ build

EnOSlib [Che+22]

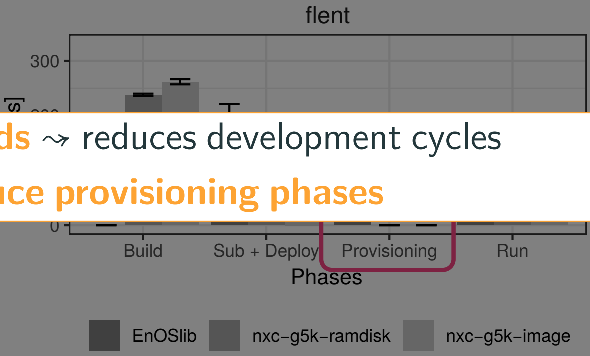


NixOS Compose ⇝ provisioning done in image

Kameleon [Rui+15]



EnOSlib [Che+22]



↪ Fast builds, **faster rebuilds** ⇝ reduces development cycles

↪ Fast deploys, **reduce provisioning phases**

NixOS Compose ⇝ provisioning done in image

build ⇝ modify (add `hello`) ⇝ build

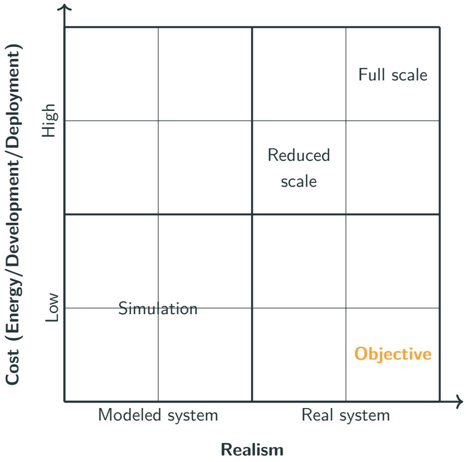
Experiment costs and reproducibility: Wrapping up

Objectives

- Reduce cost of experimenting with grid/cluster middlewares ✓
- Improve development cycles for reproducible experiments ✓

Limitations and Perspectives

- More popular Parallel File-Systems
- Source of the performance loss unclear
- Other platforms for NixOS Compose
- Hybrid/folded deployments
- Simulation: PFS and sensors



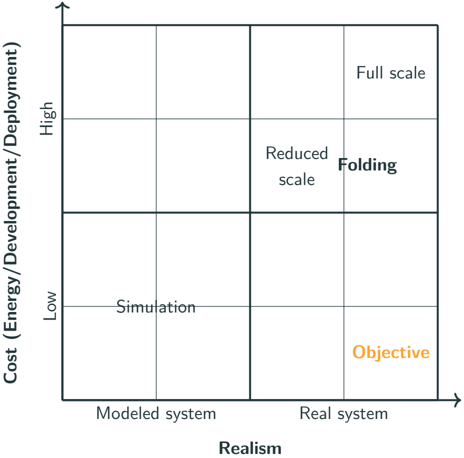
Experiment costs and reproducibility: Wrapping up

Objectives

- Reduce cost of experimenting with grid/cluster middlewares ✓
- Improve development cycles for reproducible experiments ✓

Limitations and Perspectives

- More popular Parallel File-Systems
- Source of the performance loss unclear
- Other platforms for NixOS Compose
- Hybrid/folded deployments
- Simulation: PFS and sensors



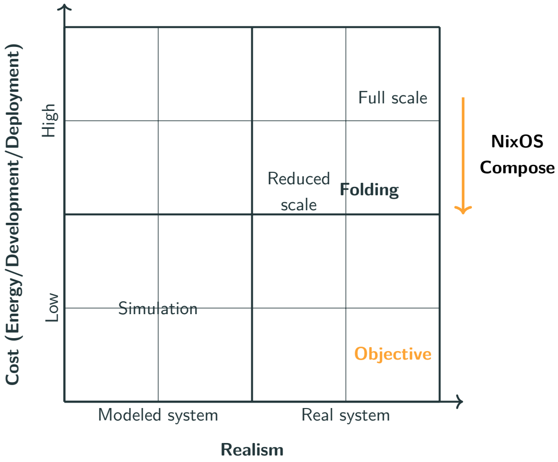
Experiment costs and reproducibility: Wrapping up

Objectives

- Reduce cost of experimenting with grid/cluster middlewares ✓
- Improve development cycles for reproducible experiments ✓

Limitations and Perspectives

- More popular Parallel File-Systems
- Source of the performance loss unclear
- Other platforms for NixOS Compose
- Hybrid/folded deployments
- Simulation: PFS and sensors



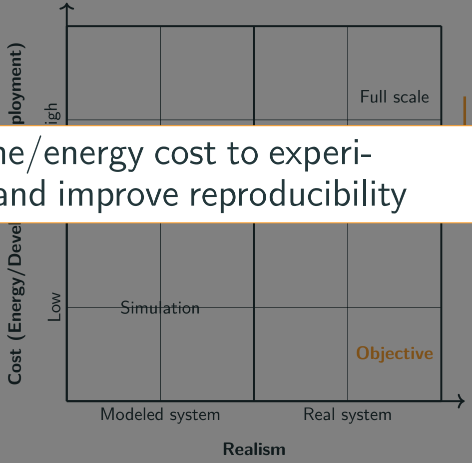
Experiment costs and reproducibility: Wrapping up

Objectives

- Reduce cost of experimenting with grid/cluster middlewares ✓
- Improve development cycles for

Take Away: Reduced the time/energy cost to experiment with distributed systems, and improve reproducibility

- More popular Parallel File-Systems
- Source of the performance loss unclear
- Other platforms for NixOS Compose
- Hybrid/folded deployments
- Simulation: PFS and sensors



Concluding thoughts

Conclusion

Initial Problem

How to harvest HPC idle resources while controlling the impact on the priority jobs?

Contributions

- Design/implement an Autonomic loop in CiGri...
  - to control the load of the File-System $\rightsquigarrow$ control overhead, avoid overload
  - to reduce the wasted computing power (idle and killed)
- ... using Control Theory
  - yields guarantees and explainability
  - guidelines for system administrators, tutorial
- Reduce experiment costs
  - reduce number of machines to deploy without loss of realism
  - tool for developing and deploying reproducible distributed environments

Open question

CiGri
Improve usage of computing clusters

Folding
Reduce number of physical machines required to represent a full scale cluster

NixOS Compose
Reduce development time, and reduce "test" deployments

Open question

CiGri
Improve usage of computing clusters

Folding
Reduce number of physical machines required to represent a full scale cluster

NixOS Compose
Reduce development time, and reduce "test" deployments

But can it introduce a **rebound effect**?

How to choose the reference value?

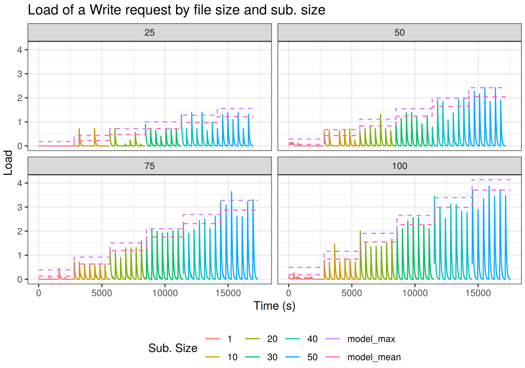
- Normalized `loadavg` then fix to 75%, 90%, 95%, etc.

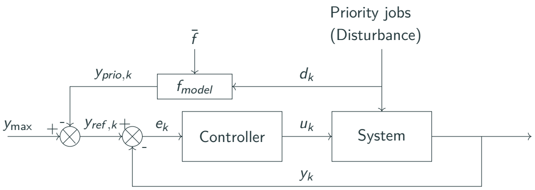
How to choose the reference value?

- Normalized `loadavg` then fix to 75%, 90%, 95%, etc.
- How much **burst** to sustain?

How to choose the reference value?

- Normalized `loadavg` then fix to 75%, 90%, 95%, etc.
- How much **burst** to sustain?
  - dynamic reference value
  - based on number of **priority jobs** and **historical I/O data** (e.g., Darshan [Car+11])



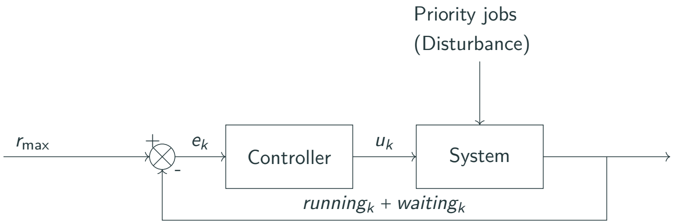


Beyond idle resources

Wasted computing power: Idle resources, but also

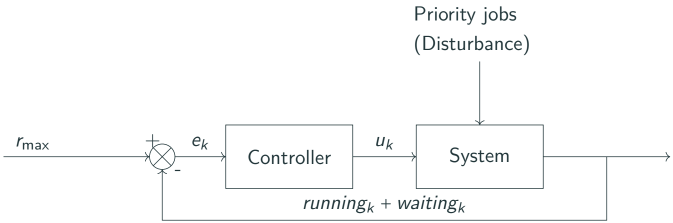
Beyond idle resources

Wasted computing power: Idle resources, but also **killed jobs**!



Beyond idle resources

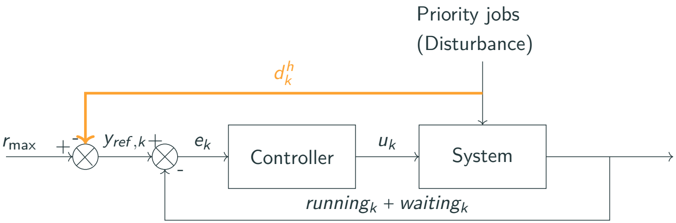
Wasted computing power: Idle resources, but also **killed jobs**!



- **anticipate** variations in available resources

Beyond idle resources

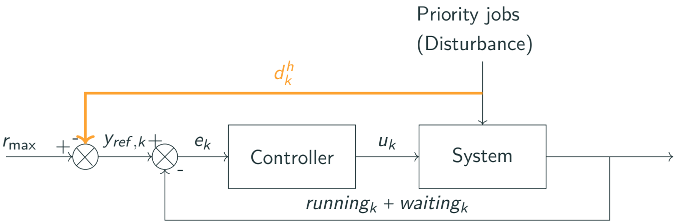
Wasted computing power: Idle resources, but also **killed jobs**!



- **anticipate** variations in available resources
- new sensor (modify OAR)
- **provisional** Gantt chart

Beyond idle resources

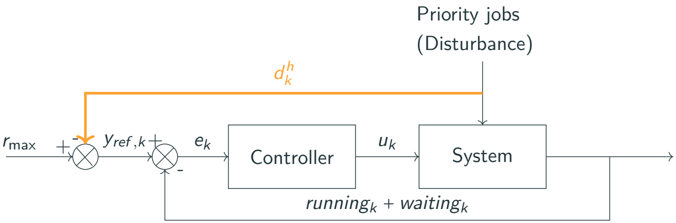
Wasted computing power: Idle resources, but also **killed jobs**!



- **anticipate** variations in available resources
- new sensor (modify OAR)
- **provisional** Gantt chart
- **horizon**

Beyond idle resources

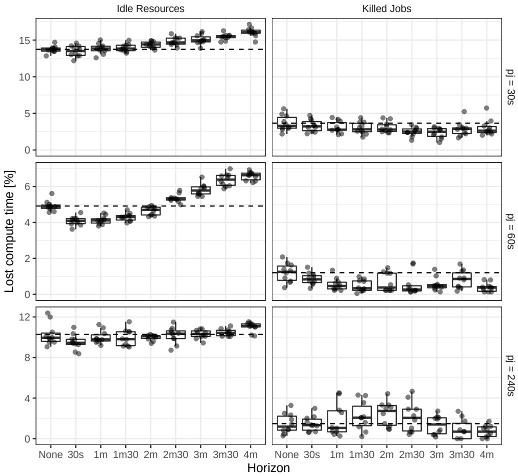
Wasted computing power: Idle resources, but also **killed jobs**!

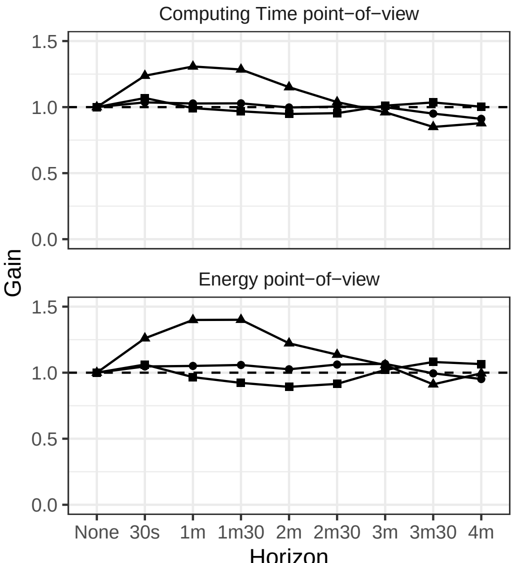


- **anticipate** variations in available resources
- new sensor (modify OAR)
- **provisional** Gantt chart
- **horizon**

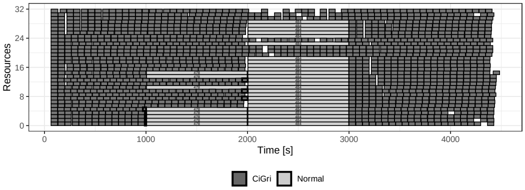
Can reduce **both idle and killed** time, and energy usage!

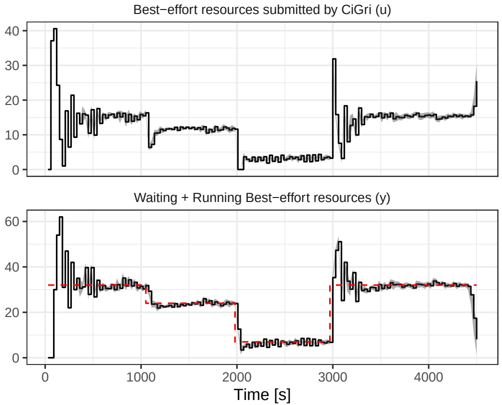
Beyond idle resources - Results





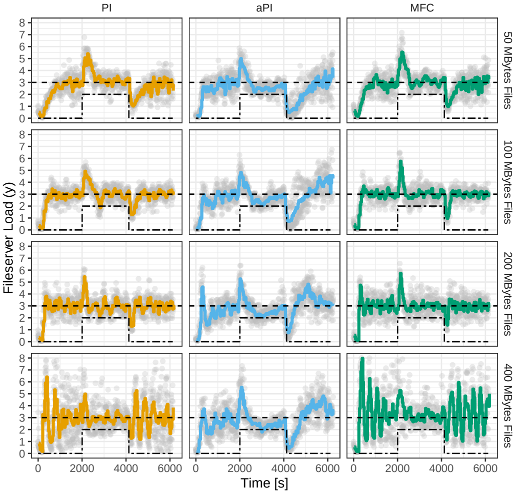
Beyond idle resources - Results



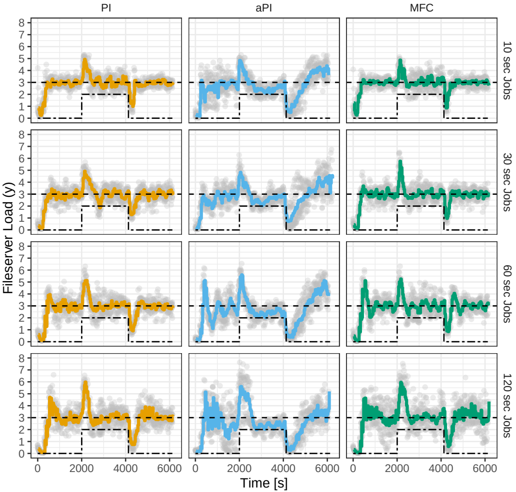


Reusability

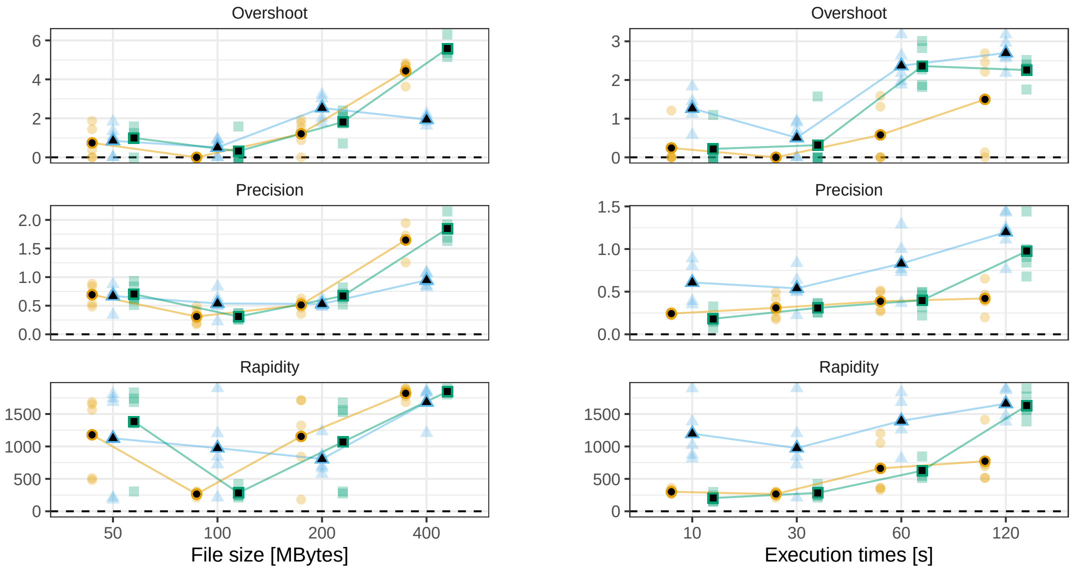
Comparison with variations in the I/O impact of jobs



Comparison with variations in the execution time of jobs



Reusability - Metrics



How to store the packages?

Usual approach: Merge them all

- Conflicts
- `PATH=/usr/bin`

```
/usr
├── bin
│   └── myprogram
└── lib
    ├── libc.so
    └── libmylib.so
```

Nix approach: Keep them separated

+ Pkg variation
+ Isolated
+ Well def. `PATH`
+ Read-only

```
/nix/store
├── y9zg6ryffgc5c9y67fcmfdkyyiivjzpj-glibc-2.27
│   └── lib
│       └── libc.so
└── nc5qbagm3wqfg2lv1gwj3r3bn88dpqr8-mypkg-0.1.0
    └── bin
        └── myprogram
    └── lib
        └── libmylib.so
```

References i

[1] R. Bleuse, "Apprehending heterogeneity at (very) large scale," Theses (Université Grenoble Alpes, Oct. 2017).

[2] J. Borrill et al., "Investigation of leading hpc i/o performance using a scientific-application derived benchmark," in Proceedings of the 2007 acm/ieee conference on supercomputing (2007), pp. 1–12.

[3] U. of California, *Ior benchmark*, 2023.

[4] P. Carns et al., "Understanding and improving computational science storage access through continuous characterization," ACM Transactions on Storage (TOS) **7**, 1–26 (2011).

[5] R.-A. Cherrueau et al., "EnosLib: A Library for Experiment-Driven Research in Distributed Computing," en, IEEE Transactions on Parallel and Distributed Systems **33**, 1464–1477 (2022).

[6] A. Déprez et al., "Toward the generation of epos-gnss products," in 19th general assembly of wegener: on earth deformation & the study of earthquakes using geodesy and geodynamics (2018).

[7] D. Ferrari et al., *An empirical investigation of load indices for load balancing applications*, (Computer Science Division, University of California, 1987).

[8] A. Filieri et al., "Software engineering meets control theory," in 2015 ieee/acm 10th international symposium on software engineering for adaptive and self-managing systems (IEEE, 2015), pp. 71–82.

References iii

[9] Y. Georgiou et al., "Evaluations of the lightweight grid cigri upon the grid5000 platform," in Third ieee international conference on e-science and grid computing (e-science 2007) (IEEE, 2007), pp. 279–286.

[10] Q. Guilloteau et al., "Étude des applications bag-of-tasks du méso-centre gricad," in COMPAS 2022 - Conférence francophone d'informatique en Parallélisme, Architecture et Système (July 2022), pp. 1–7.

[11] Q. Guilloteau et al., "Folding a Cluster containing a Distributed File-System," working paper or preprint, 2023.

References iv

[12] Q. Guilloteau et al., "Painless Transposition of Reproducible Distributed Environments with NixOS Compose," in CLUSTER 2022 - IEEE International Conference on Cluster Computing, Vol. CLUSTER 2022 - IEEE International Conference on Cluster Computing (Sept. 2022), pp. 1–12.

[13] M. Imbert et al., "Using the EXECO toolbox to perform automatic and reproducible cloud experiments," in 1st International Workshop on UsiNg and building ClOud Testbeds (UNICO, collocated with IEEE CloudCom 2013 (Dec. 2013).

[14] J. O. Kephart et al., "The vision of autonomic computing," Computer **36**, 41–50 (2003).

References v

[15] M. Mercier et al., "Big data and hpc collocation: using hpc idle resources for big data analytics," in 2017 ieee international conference on big data (big data) (IEEE, 2017), pp. 347–352.

[16] B. Przybylski et al., "Using unused: non-invasive dynamic faas infrastructure with hpc-whisk," in Sc22: international conference for high performance computing, networking, storage and analysis (IEEE, 2022), pp. 1–15.

[17] C. Ruiz et al., "Reconstructable Software Appliances with Kameleon," en, ACM SIGOPS Operating Systems Review **49**, 80–89 (2015).