Minimizing Cluster under-use with a Control-based approach

Quentin Guilloteau

Thursday 25th June, 2020

Supervisors: Olivier RICHARD (DATAMOVE) & Eric RUTTEN (CTRL-A)
In Collaboration with Gipsa Lab

Introduction

High Performance Computing (HPC)

HPC

HPC systems are more and more complex:

- unpredictable in runtime performances
- unpredictable in energy consumption
- unpredictable in data access time

Potential Problems

- Overloading
- Overheating

$\hookrightarrow$ need runtime management to meet performance objectives

A Possible Solution

Autonomic Computing

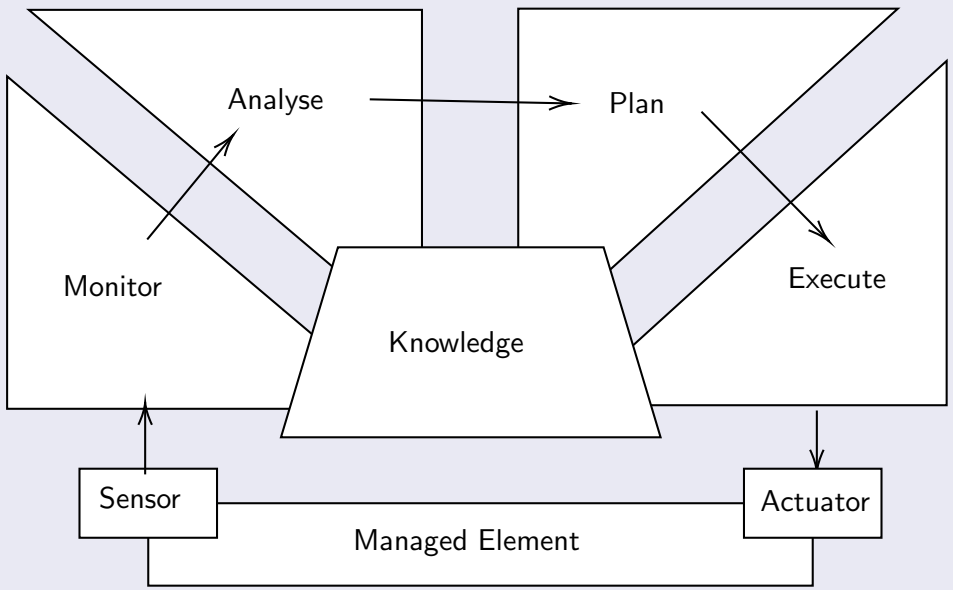
Autonomic Computing

Definition

Systems that can **manage themselves** given **high-level objectives** from administrators

Main tool: the MAPE-K Loop

1. **M**onitor
2. **A**nalyse
3. **P**lan
4. **E**xecute
5. **K**nowledge

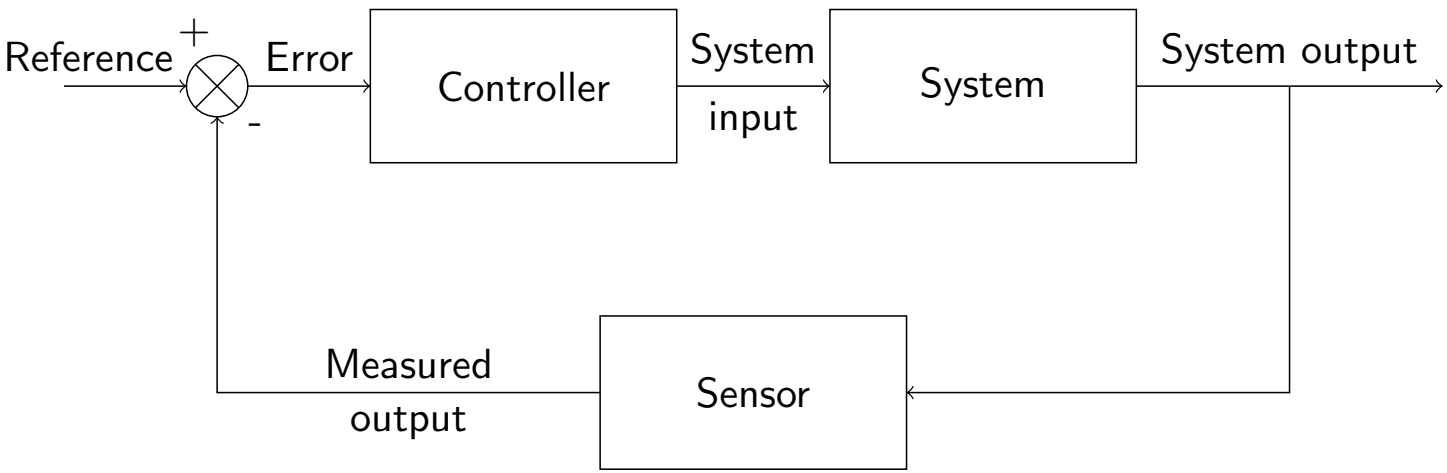


Control Theory

Control Theory

Manages the control of continuously dynamical systems
(i.e. make systems behave in a desired way)

↪ Interpretation of MAPE-K loop



Challenges

↪ Control Theory requires models, testing controllers, etc

Reproducibility

Definition

Make Science:

- **Repeatable** (same experiment, same results)
- **Replicable** (same experiment with different input)

↪ More importantly, **Verification** & **Reusability**

In the context of this Project

Control Theory models based on experimental results
↪ **Requires experiments of quality**

Outline

1 Introduction

2 The CiGri Middleware

3 Minimize Cluster under-use while regulating the Load

4 The Quest of Reproducibility: Transposition

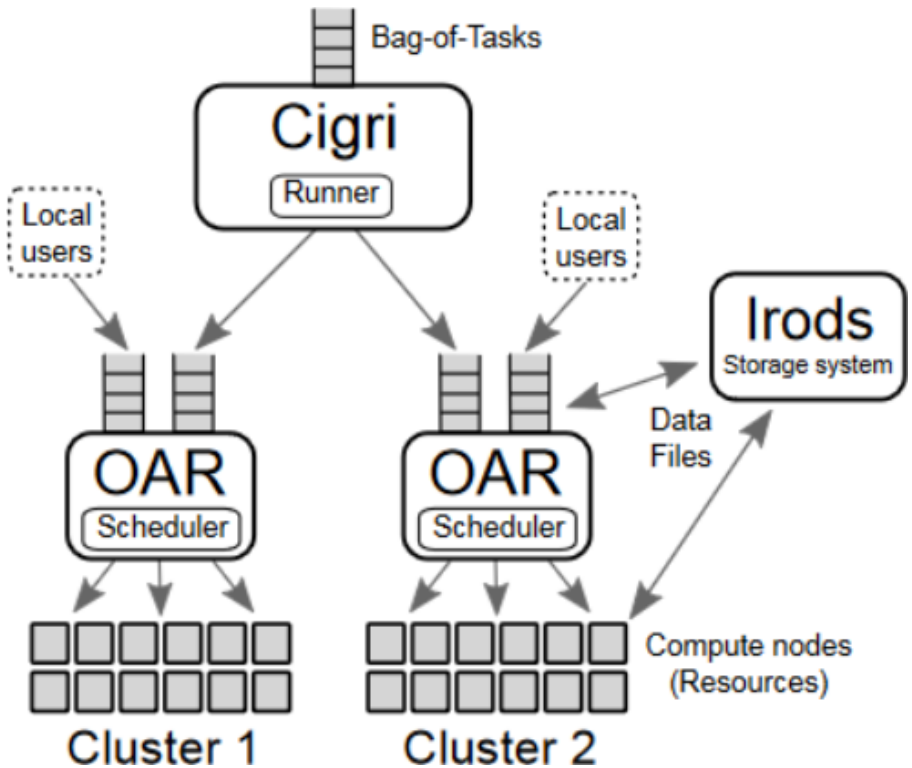
5 Conclusion

The CiGri Middleware

CiGri - Presentation

CiGri (CIMENT Grid)

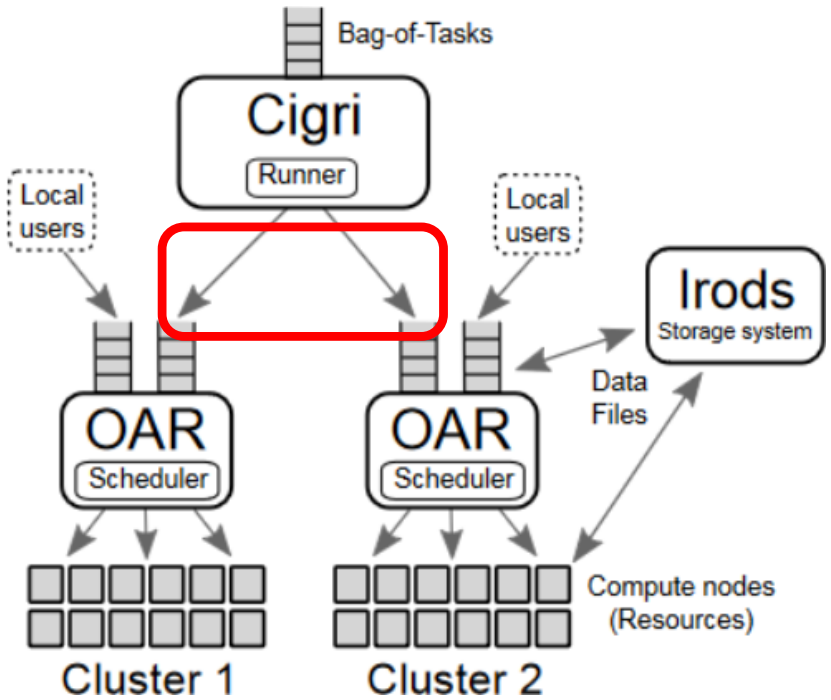
- Lightweight fault-tolerant grid middleware application
- Runs on top of a set of OAR clusters
- **Goal**: Exploit unused resources of a cluster
- **bag-of-tasks**: Large sets of multi-parametric tasks
- **Best-effort Jobs**: Jobs with lowest priority



CiGri - Submission Loop (1/2)

Algorithm 1: Submission Loop

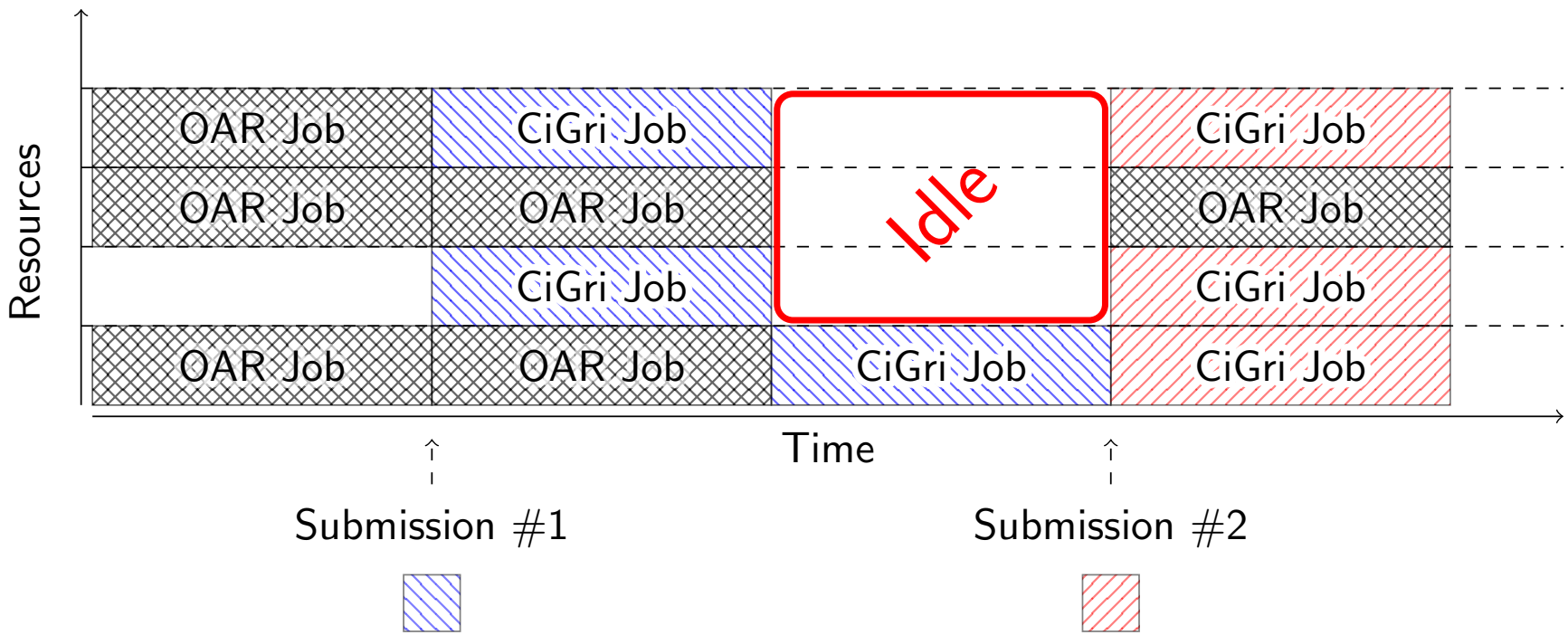
```
rate = 3;
increase_factor = 1.5;
while jobs left in bag-of-tasks
  do
    if no running jobs then
        launch rate jobs;
        rate = min(rate ×
          increase_factor, 100);
    end
    while jobs running > 0 do
        sleep until timeout;
    end
end
```



CiGri - Submission Loop (2/2)

Issue

Must wait for completion of previous submission to submit again
↪ reduce **overload** but can lead to **under-utilisation**



CiGri - The need for improvement

Observation

Current CiGri algo too protective
$\hookrightarrow$ could be improved if takes into account state of the cluster

Idea

Regulate the number of jobs submitted to OAR w.r.t. number of idle resources in the cluster
Add constraints (e.g. Load of the fileserver)

State-of-the-art / Previous Work

Applied Autonomic Computing to CiGri

Minimize Cluster under-use while regulating the Load

Introduction & Previous Work

Objective

Submit jobs to OAR while keeping load of fileserver under a given value

Model Based Controller

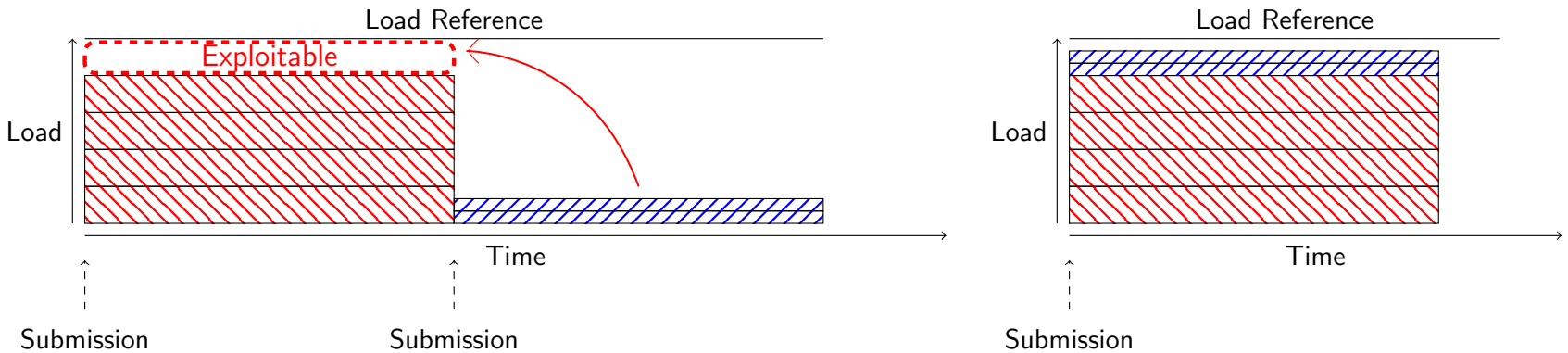
- Use models of the system to make prediction
- Conclude the best number of jobs to send

Introduction & Shortfall

Problem

Originally, CiGri submission to OAR composed of jobs from same campaign, thus same behaviour

$\hookrightarrow$ could lead to some under-utilisation



Use case

Submitting jobs from two campaigns with different IO loads

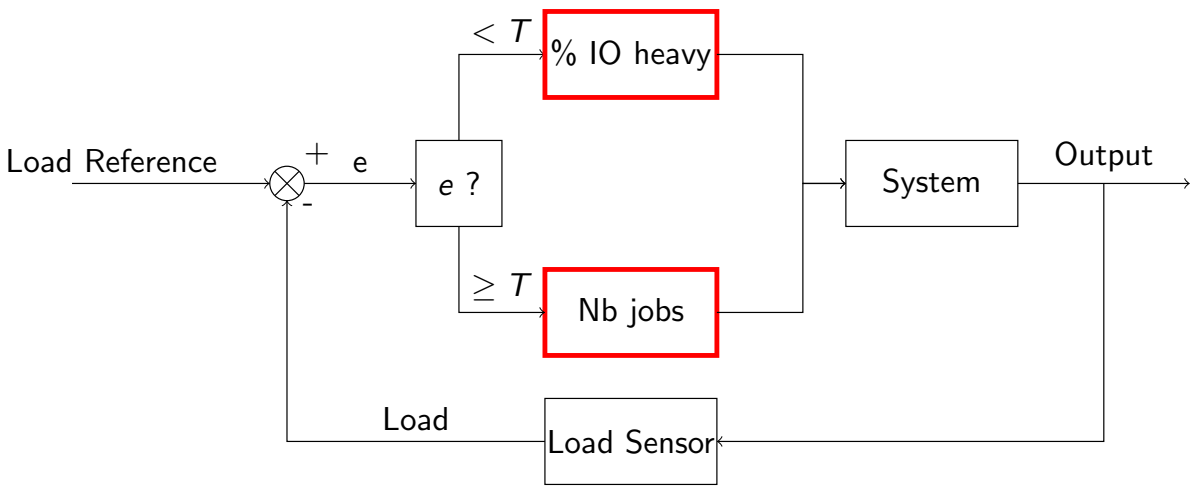
Proposed Controller

Presentation: Two modes of control

1. The **total number of jobs** sumbitted to OAR ("big step")
2. The **percentage of IO heavy jobs** submitted ("small step")

Proportional Controller: Response proportional to the Error

$$Output = \mathbf{k} \times Error = \mathbf{k} \times (Ref - Load)$$



Experimental Setup

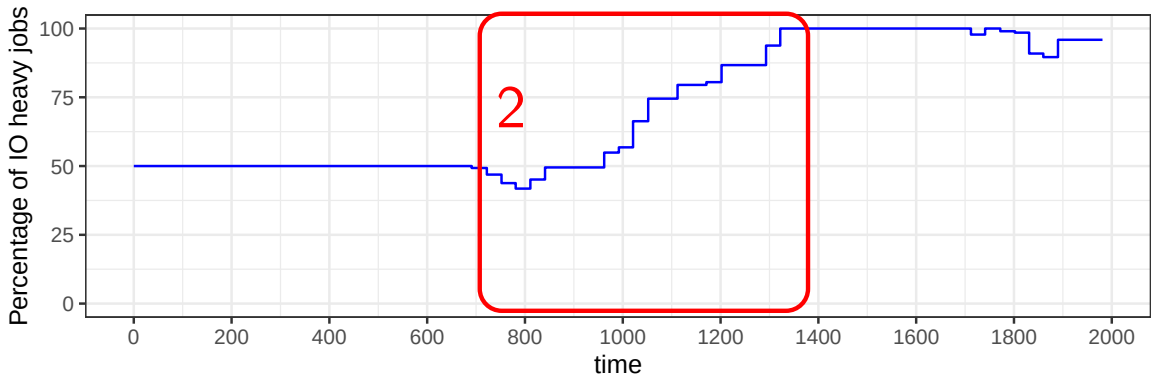
Architecture

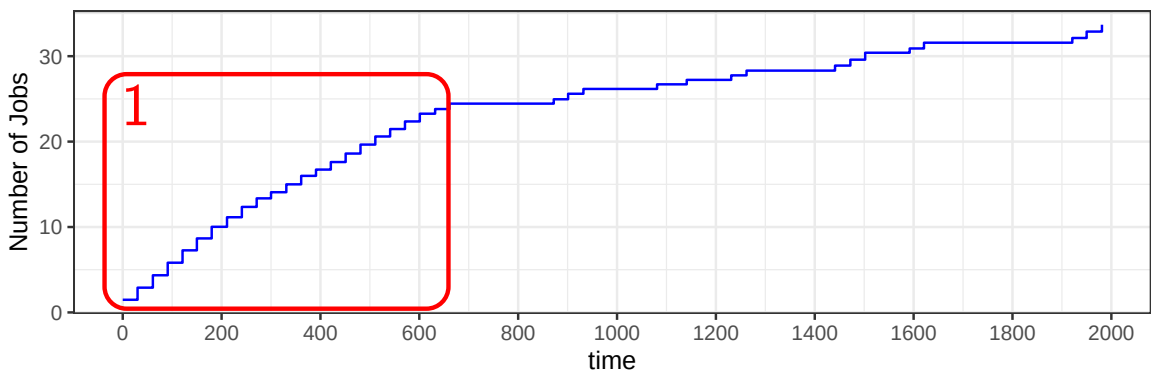
- Grid5000 (Nancy Grisou: 2 Intel Xeon E5-2630v3, 8 cores/CPU, 128 GiB)
- 1 CiGri (v3) server
- 1 OAR (v3) server
- 1 Fileserver (NFS)
- 100 nodes

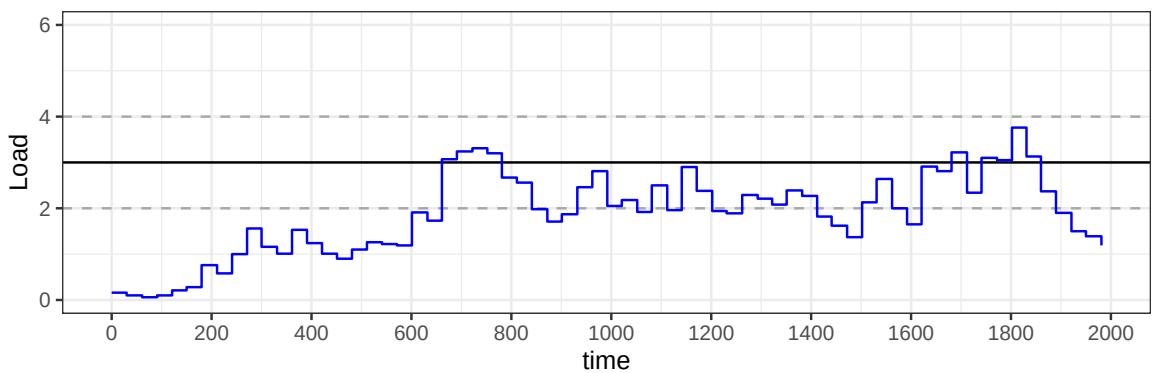
Experiment

- IO heavy campaign: 1000 jobs: sleep 30s, writes 10 MB file
- IO light campaign: 1000 jobs: sleep 30s
- Regulate load around value **3** (chosen by admin.)
- Threshold value of **1**

Experimental Results







Takeaways

1. Rising phase
2. then small steps
3. Kept load in interval
4. Kept load mostly $<$ Ref

Discussion on the Control Theory Approach

Results

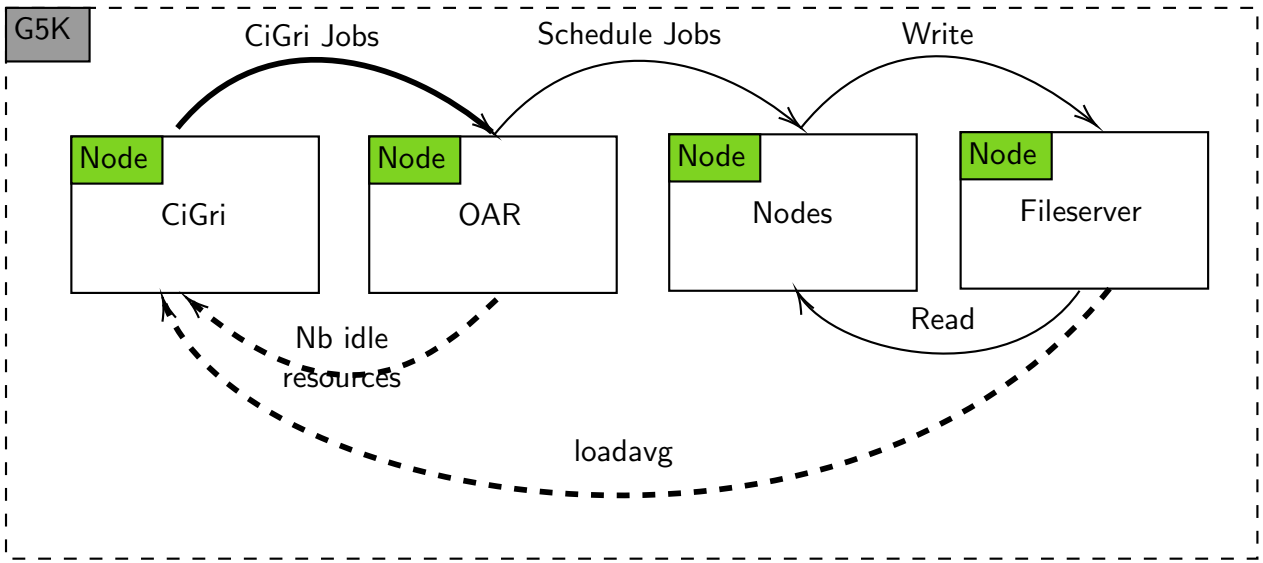
Improved cluster utilisation compared to model based approach:
25% vs. 6%
(Difficult to have precise models)
(Utilisation depends on chosen reference value)

Future Work

- proof-of-concept
- need further testing
- try with more complex controllers
- different ways to switch between modes
- how to categorize campaigns (heavy or light) ?

The Quest of Reproducibility: Transposition

Transposition: Motivation



Motivation

- Not everyone has access to multiple machines
- Long deployment time ($\simeq$ 15 minutes)
- Long development time

$\hookrightarrow$ Using a **container** approach: faster & lightweight

Transposition: Definition & Questions

Definition

Transpose a system to a different platform **while keeping guarantees on the behaviour**

↪ Transpose **feedback loops** and **sensors** to containers

Scientific Questions

Are the experiments (distributed & container) **comparable** ?

Can we **learn** enough **about the system** with containers ?

Can we **develop new models/controllers, meaningful on all platforms** ?

↪ Study with **Nix**

Nix: a Tool to improve Reproducibilty

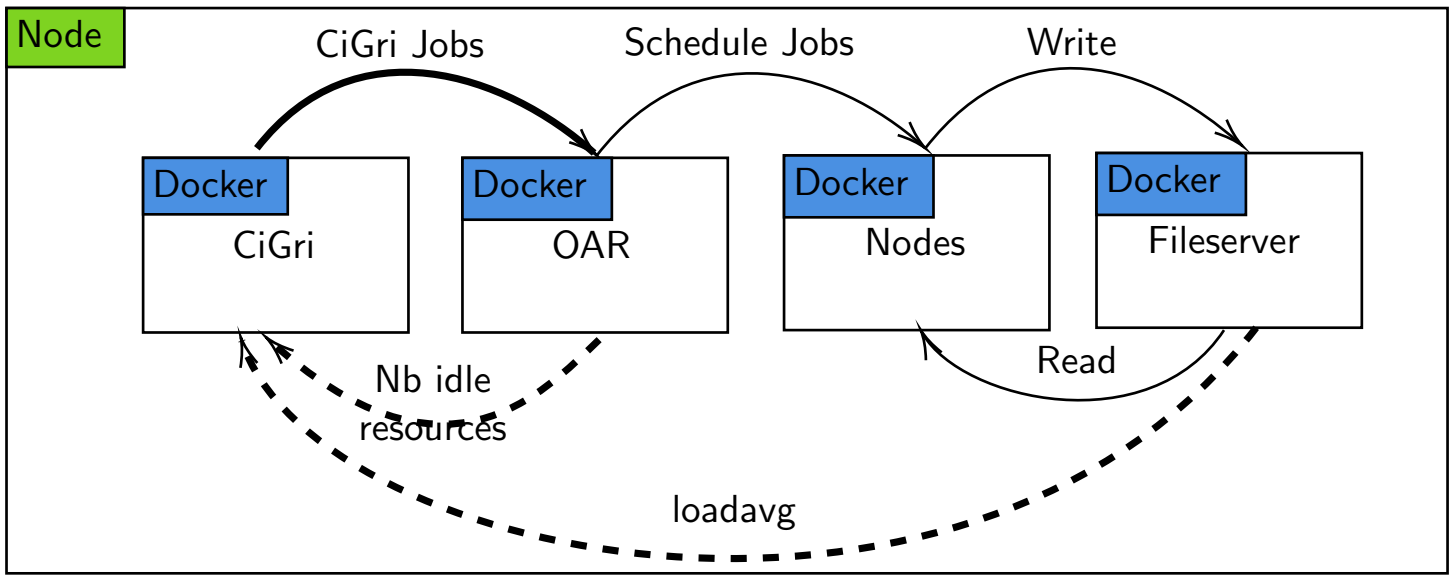
Nix

- Functional Package Manager
- Nix language: declarative **def. of complete software stack**
- Reproducible: **Tracability of builds**
- Reliable: roll back & cannot break other packages

```
{ stdenv, fetchurl, perl }:

stdenv.mkDerivation {
  name = "hello-2.1.1";
  builder = ./builder.sh;
  src = fetchurl {
    url = "ftp://.../hello-2.1.1.tar.gz";
    sha256 = "1md7...";
  };
  inherit perl;
}
```

Arion = Nix + `docker-compose`



Arion

Nix wrapper around `docker-compose`

Arion - Common Configuration (Snippet)

```
service.volumes = [ "${builtins.getEnv "PWD"}/..:/srv" ];
service.capabilities = { SYS_ADMIN = true; }; # for nfs
service.useHostStore = true;
nixos.configuration = {
  networking.firewall.enable = false;
  boot.tmpOnTmpfs = true;
  users.users.user1 = {isNormalUser = true;};

  environment.systemPackages = with pkgs; [ nfs-utils socat wget ruby openssh ];
  imports = lib.attrValues pkgs.nur.repos.kapack.modules;

  environment.etc."privkey.snakeoil" = { mode = "0600"; source = snakeOilPrivateKey; };
  environment.etc."pubkey.snakeoil" = { mode = "0600"; source = snakeOilPublicKey; };

  services.oar = {
    database = {
      host = "server";
      passwordFile = "/srv/common/oar-dbpassword";
    };
    server.host = "server";
    privateKeyFile = "/etc/privkey.snakeoil";
    publicKeyFile = "/etc/pubkey.snakeoil";
  };
};
```

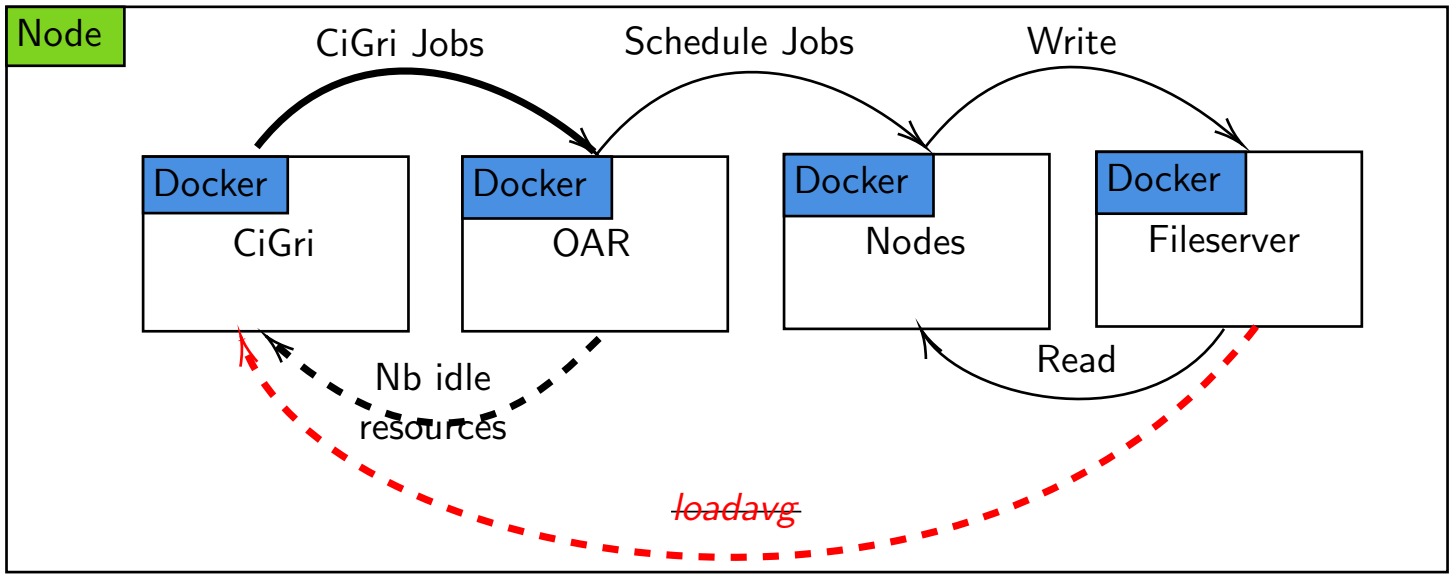
Figure: Arion: Common configuration of a node

Arion - Node Configuration (Snippet)

```
services.fileserver = addCommon {
  service.hostname="fileserver";
  nixos.configuration = {
    services.nfs.server.enable = true;
    services.nfs.server.exports = ''/srv/shared *(rw,sync,...)'';
  };
};
```

```
services.node1 = addCommon {
  service.hostname="node1";
  nixos.configuration = {
    services.oar.node = {
      enable = true;
      register = {
        enable = true;
        extraCommand = ''
          /srv/common/prepare_oar_cgroup.sh init
          mkdir -p /mnt/shared
          mount -t nfs fileserver:/srv/shared /mnt/shared -o nolock
        '';
        nbResources = "100";
      };
    };
  };
};
```

Architecture - Containers & new Problem



Problem

Sensor `/proc/loadavg` **not available** in container

↪ How to transpose the feedback loop ?

`loadavg` - Computation by Hand in a Container

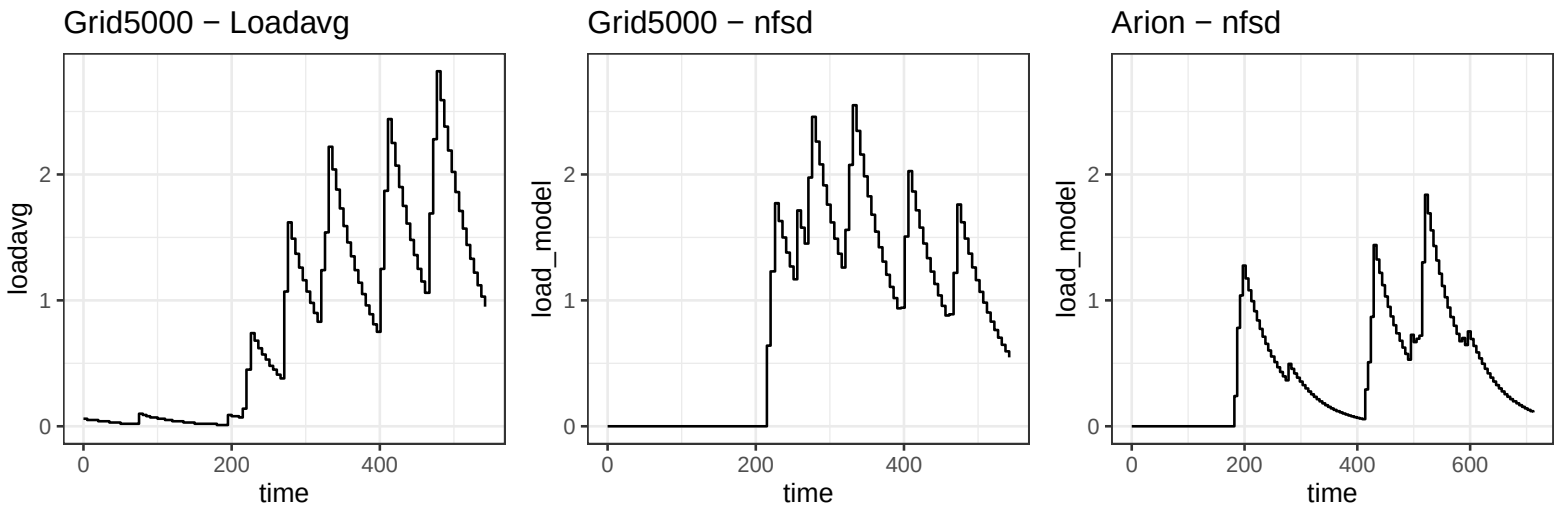
`loadavg`

Number of jobs in the running queue or waiting for the disk

↪ Processes on fileserver: `nfsd` processes (from NFS server)
↪ Compute `loadavg` by counting `nfsd` processes

New Problem

NFS Server → Kernel processes: not visible inside container
↪ but visible from the host



Discussion

Lessons Learned

- Faster Deploy. & Dev. w/ Nix and Arion (1 min vs. 15 mins)
- Transpo.: Not looking for identical behaviour, but similar for some priorities

Open Questions

- Change the metric (pseudo-`loadavg` or new one) ?
- Same models (for Arion & G5K) with different parameters ?
- Impact of a different sensor for the design of a controller ?

Conclusion

Conclusion & Contributions

Conclusion

- New controller for minimizing cluster under-use while regulating the load of the fileserver
- Transposition to a container approach

Contibutions

- Designed and Implemented Controller to improve granularity
- Investigation on Transposition
- Runnable Lab Notebooks w/ OrgMode (collab. Gipsa Lab)

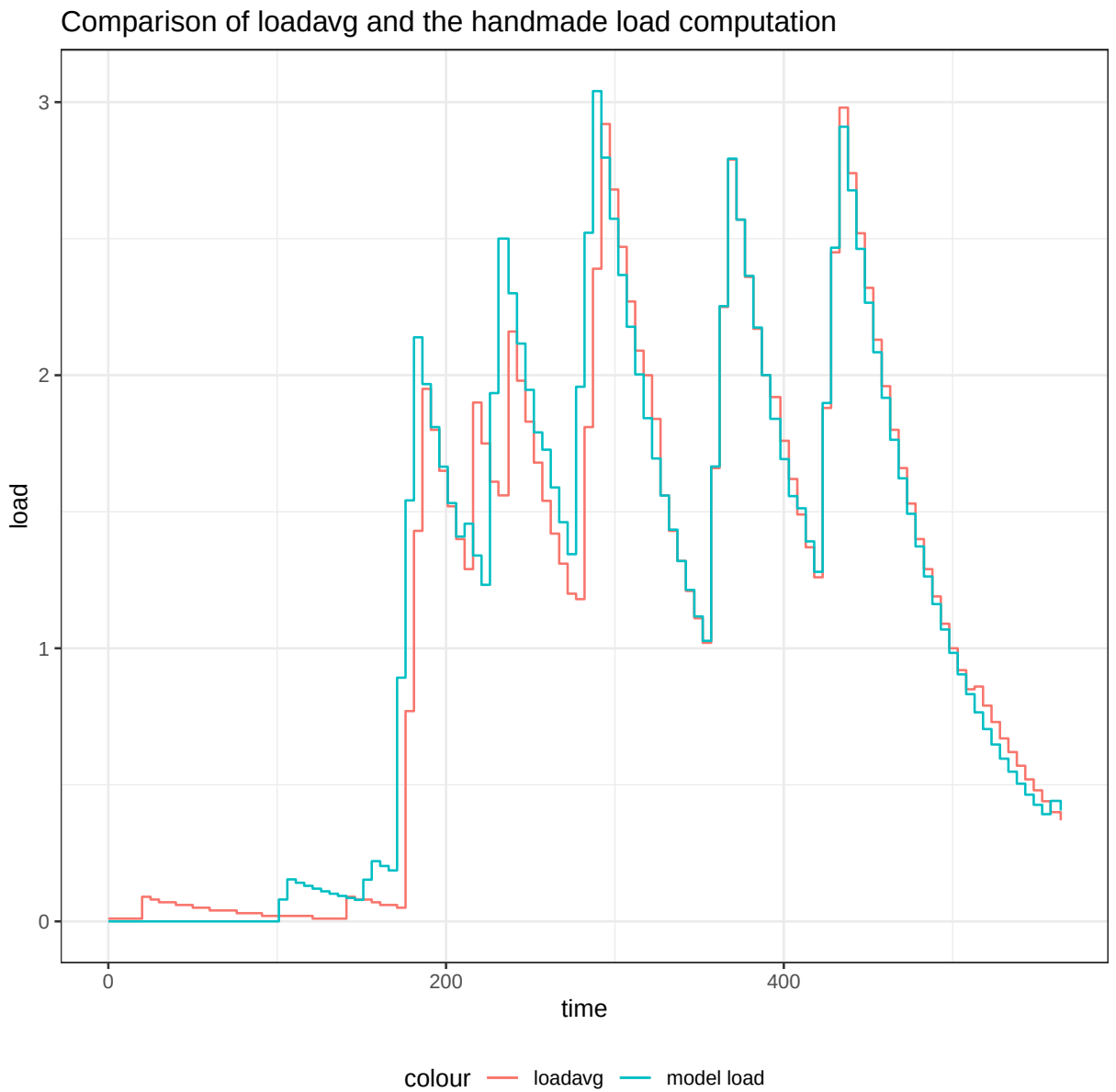
Future Work

- Continue the transposition to containers
- Tests controllers on different types of workloads
- Nix + Grid5000 = to complete
- Control number of jobs submitted w.r.t. other metrics (e.g. energy consumption)

Questions ?

Thank you for your attention.
Time for Questions !

loadavg - Computation by Hand on G5K & Validation



`loadavg` - a Quick Presentation

Definition

- Output of `/proc/loadavg`
- Represents the number of jobs in the running queue or waiting for the disk
- Computed as an exponential weighted sum of the number of jobs

$$f_i = \left(1 - e^{-T}\right) f_{i-1} + n_i e^{-T}$$

Why is it an interesting metric ?

Provides a **sense of overload**

$\hookrightarrow$ Rule-of-thumb: if `loadavg` $>$ number of cores, then overload