Controlling the Injection of Best-Effort Tasks to Harvest Idle Computing Grid Resources

ICSTCC 2021, Iași

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

*Université Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG

**Université Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab

2021-10-21

Did you say HPC ?

High Performance Computing

Process data and perform complex computations at high speed

Cluster

Set of (near) identical machines, linked by network, share storage

↪ Grid = set of clusters

Usual workflow

1. User reserves nodes/machines in cluster
2. Scheduler assigns machines to the user's job
3. Once machines ready, user executes job on machines
4. Job can do some I/O
5. Once job over, machines are freed and ready for another user

Harvesting of idle resources

Scheduling leaves "holes"

- specifications
- wrong estimation of exec. time
- ...

Objective

Use the idle resources to execute low priority jobs

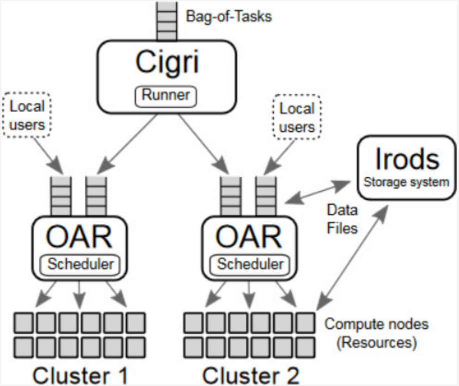
State of the Art

- BOINC/Condor: gathers CPU cycles from idle *personal* machines
- OurGrid: sharing of machines between $\neq$ labs
- BeBiDa: BigData jobs on idle HPC machines

CiGri: Presentation

CiGri (CIMENT Grid)

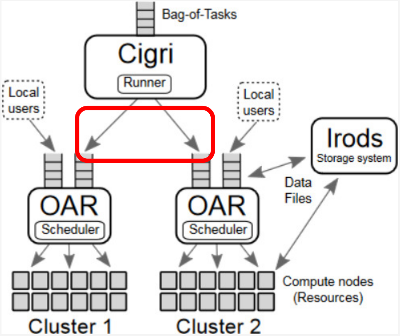
- Fault tolerant middleware
- Interact with a set of (*OAR*) schedulers
- **Goal**: Exploit idle resources of a grid in a **non-intrusive way**
- **bag-of-tasks**: Large set of multi-parametric tasks
- **Task Best-effort**: Task with the **lowest** priority



CiGri: Submission Loop (1/2)

Algorithm 1: Current Solution

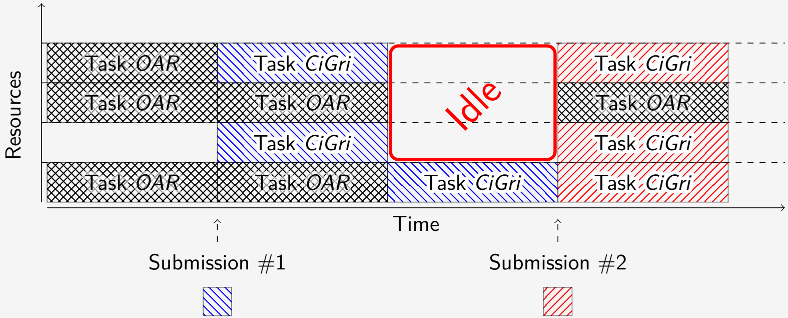
$rate = 3$;
$increase\_factor = 1.5$;
while *tasks not executed in b-o-t* **do**
  if *no task running* **then**
    submit *rate* tasks;
    $rate = \min(rate \times increase\_factor, 100)$;
  end
  while *nb of tasks running > 0* **do**
    sleep during 30 sec;
  end
end



CiGri: Submission (2/2)

The Issue

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



↪ take into account current state of cluster (resources + DFS)

↪ **use Control Theory tools**

Distributed File System and its Sensor

Distributed File System (or Fileserver)

stores the files of the cluster's users

Potential Issue

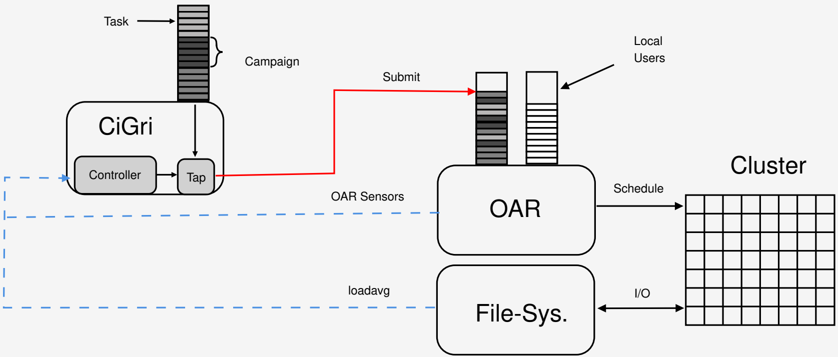
Too much simultaneous reads/writes $\Longrightarrow \nearrow$ read/write time
$\hookrightarrow$ perturbations of premium users jobs $\Longrightarrow$ ☹ (**non-intrusive !**)

$\hookrightarrow$ How to sens the overload of the DFS ?

Our sensor: `/proc/loadavg`

- Represents the number of CPU processes running or waiting for disk
- Running avg. window $\rightarrow$ Inertia

CiGri: its feedback loops, sensors and actuators



Definition of the Problem and Previous Work

Objective

Use as much idle resources from the cluster **without overloading** the Fileserver

$\hookrightarrow$ reference value for the load of the DFS
$\hookrightarrow$ controller on this reference value

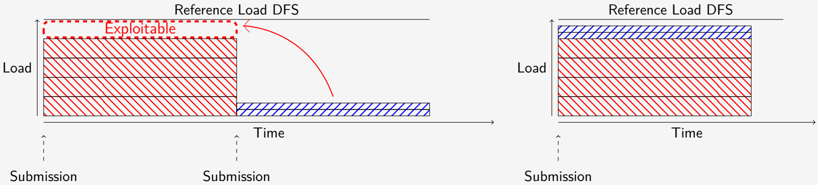
Previous Work

- PI Controller to regulate the quantity of tasks to submit:
  - improved cluster usage vs. original solution
  - ~~File Server~~
- MPC Controller taking into account DFS:
  - Too simple model
  - Did not scale

Observation

Problem

Originally: *CiGri* subs composed of tasks from the **same campaign**
$\hookrightarrow$ $\simeq$ same behaviour (exec. time + I/O)
$\hookrightarrow$ can lead to cluster under-utilisation



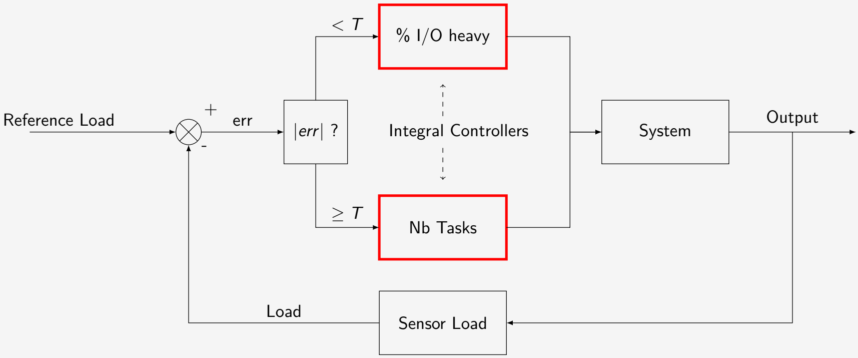
Strategy

Submit tasks from 2 $\neq$ campaigns with $\neq$ I/O loads

Proposed Regulation: Bisphasic Approach

Two control modes/phases

1. **total number of tasks** submitted to *OAR* ("big step")
2. **percentage of tasks I/O heavy** submitted ("small step")



Experimental Setup

Architecture

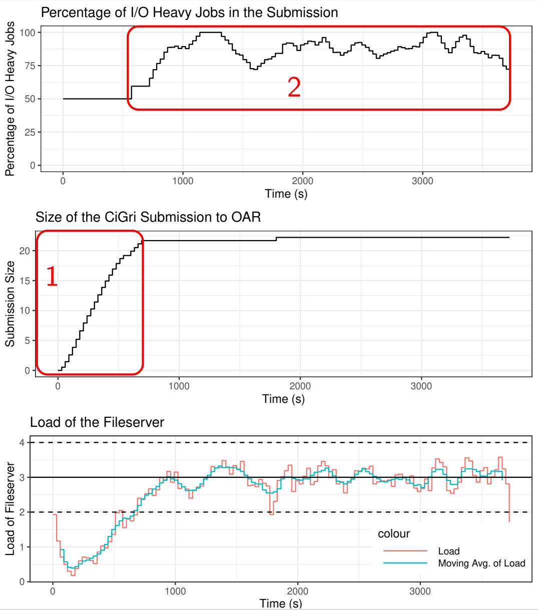
4 nodes from Grid'5000 (Grisou)

- 1 Server *OAR* (v3)
- 1 Server *CiGri*
- 1 Distributed File System (NFS)
- Emulation of a 100 nodes cluster

Experiment

- I/O heavy Campaign of 1000 tasks: `sleep` 30s + write 100MBytes
- I/O light Campaign of 1000 tasks: `sleep` 30s + write 10MBytes
- Load Reference = 3, Threshold = 1
- Premium Users: synthetic behaviour

Results without premium users



Remarks

1. Rising phase("big steps")
2. Then "small steps"
3. Keeps load in interval

Results with premium users

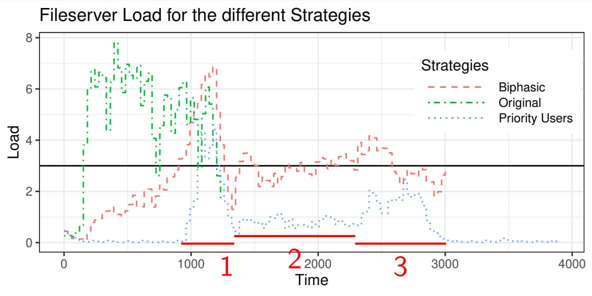


Figure: DFS Load with Premium Users

Remarks

- Synthetic Load (3 Phases)
  - Slow reaction to variations
  - Otherwise, manage to keep load in interval

Comparison

Reminder of the Objective

Using more idle resources of the cluster **without overloading** the DFS

Strategy	Feedback	Proportion of time spent w/ DFS overloaded (%)			Cluster Usage (%)	
		Load > Ref	Load > Ref + 33%	Load > Ref + 66%	absence	presence
Original	✗	80.49%	75.61%	60.98%	63.7	73.8
Scan	✓	17.86%	7.14%	3.57%	13.8	16.1
MPC	✓	89.02%	84.15%	59.76%	28.2	44.5
Simple Control	✓	25.93%	11.11%	7.40%	17.4	23.0
Biphasic Control	✓	37%	10%	7%	19.6	25.8

Remarks

- Good load regulation for Biphasic (Top 3)
- Best cluster usage within this Top 3

- Best usage vs. simple control → usefulness of Biphasic

Conclusion & Perspectives

Conclusion

- Solution managing to regulate the DFS load around reference value
- while harvesting more idle resources

Perspectives

- Relation between `loadavg` and read/write times
- Automatic detection of campaign type (I/O heavy or I/O light)
- Real load for premium users
- Identification for gains of controllers
- others control strategies (PID, RST, etc)