Simulating a Multi-Layered Grid Middleware

Quentin Guilloteau

Simulating a Multi-Layered Grid Middleware

Quentin Guilloteau

Univ. Grenoble Alpes, Inria, CNRS, LIG, F-38000 Grenoble France*

Abstract

The study of grid or cluster middlewares is complex, and experiments on such systems are costly. This cost can come from the number of resources required to deploy realistic experiments, or the time to replay a significative workload. Simulation techniques can help reduce such cost to almost none and help perform preliminary study at low cost, but they might also degrade the realism of the results. In this paper, we consider the implementation in simulation of the *CiGri* grid middleware, in *Batsim*, a batch sheduler simulator. We are particularly interested in the impact of simulation on signals of interest and their dynamics compared to the real system in the optic of using this simulator to accelerate the first steps of future studies.

1 Introduction

Distributed experiments are complex and often require several machines for several hours or days. Such experiments are time and resource consuming, but are nevertheless mandatory to validate research work. Deploying and running long-lasting experiments during the exploring phases of research is an obstacle to careful and sane work and must be addressed.

Simulation techniques are an adequate solution as they allow users to execute in reasonable time and on a single laptop, experiments that would have taken hours on a production platform. In the context of High-Performance Computing (HPC), most of the effort in terms of simulators is focused on tools to evaluate scheduling algorithms [4, 5, 12]. These solutions reduce considerably the time and computing power required to replay long scientific workloads with a new scheduling strategy instead of deploying a modified batch scheduler and re-executing the jobs of the workload, but have limitations in terms of realism due the underlying models.

Experiments on systems such a grid or cluster middlewares are also victim of high experimental costs and could benefit from simulation techniques. However, due to this additional layer, the simulators cited above are not directly equipped to simulate such systems.

In this paper, we present and evaluate *BatCiGri*, a simulator of the *CiGri* grid middleware within *Batsim*. Section 2 details the studied middleware and the desired properties of its simulation. In Section 3 we present the design of the simulation of the middleware as well as its calibration to better match reality. The evaluation of the simulation is performed in Section 4.

2 Motivating Example: the *CiGri* middleware

2.1 Presentation

CiGri [6] is a grid middleware in production at the French *Gricad* meso-center [1]. The goal of *CiGri* is to use the idle resources of the meso-center. It interacts with several clusters managed by *OAR* [2] batch schedulers.

Users of *CiGri* submit *Bag-of-Tasks* applications to the middleware. Such applications are composed of thousands of short, independent and similar tasks are classified as *embarrassingly parallel* which make them good candidates for "filling the holes" in the cluster schedules. Monte-Carlo simulations or parameter sweeps are examples of *Bag-of-Tasks* applications.

Once the set of tasks submitted to *CiGri*, the middleware will submit sub sets of jobs to the different schedulers of the grid. The jobs are submitted with the lowest priority (best-effort) in order to allow premium users of the clusters to get the resources used by *CiGri* jobs if needed.

Figure 1 depicts the interactions between *CiGri* and different clusters of the grid.

*Firstname.Lastname@inria.fr

[1] https://gricad.univ-grenoble-alpes.fr/index_en.html

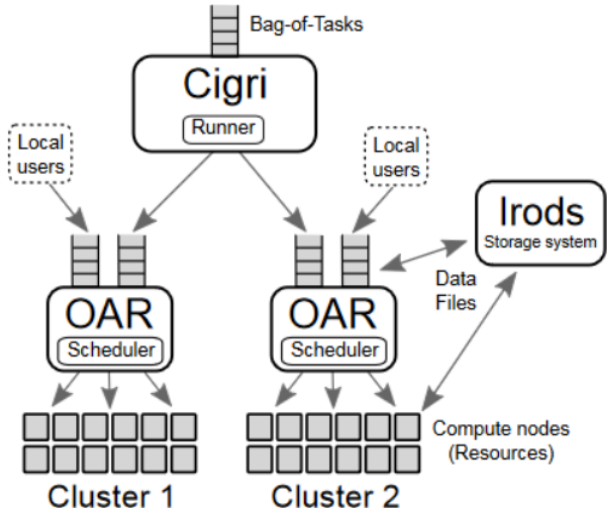


Figure 1: Interactions between *CiGri* and the schedulers *OAR* of the computing grid. *CiGri* users submit *Bag-of-Tasks* applications, whose jobs are then submitted to the different cluster schedulers of the computing grid. The computing clusters are shared with users with more priority, thus *CiGri* jobs must be killed if one premium user requires the resources.

2.2 Limitation of *CiGri*

One problem of *CiGri* is its submission algorithm. *CiGri* will submit a batch of jobs the one scheduler, and wait for the completion of the batch to submit again. This strategy can lead to an underutilization of the cluster resources. For example, *CiGri* might wait for the very last job of the previous submitted batch to terminate while there would be plenty of idle resources. Moreover, one objective of *CiGri* is to harvest in a *non-invasive* fashion. Meaning that the premium users of the different clusters must not notice the impact of the *CiGri* jobs on the platform. However, once executing, the *CiGri* jobs are using the shared resources of the cluster (file-system, network, etc.), which can have an impact on the performance of every other running jobs.

2.3 Feedback Loop Regulation

We address these limitations of the current *CiGri* submission algorithm by considering the problem from the point of view of Autonomic Computing [11]. One aspect of Autonomic Computing is the self-regulation of systems, where the controlled systems is cyclically monitored via sensors, and then based on the sensors, the autonomic controller will act on the system to direct it towards a desired state. The implementation of the decision process can be done via multiple techniques (IA, rules, modelisation and optimal solving, etc.). But in our work [8, 9], we implement the autonomic controller with tools from Control Theory. Control Theory is a field from physical engineering for the regulation of dynamic systems. It has been used for centuries on physical systems, and its properties have been proven mathematically. Its usage on computing systems is only recent. To implement a controller with Control Theory tools, the definition of the signals as well as their dynamic must be clearly identified and modeled.

To test our version of *CiGri* with our controllers, we deploy a modified *CiGri* as well as an instance of *OAR* and compute nodes. In order to perform faithful evaluations, it is unreasonable to deploy on the *entire Gricad* meso-center, and replay long workloads. We are thus interested in simulation techniques to reduce the experimental costs.

However, one potential limitation of using simulation techniques in our case, is the inability to obtain the same signals, or for the signals to have a different dynamic or properties.

2.4 Expected Properties of the Simulation

For the *CiGri* simulation to be useful from the point of view of Control Theory it must have the following properties:

- Jobs must have realistic execution times
- Best-effort jobs must be killed and release resources for the normal jobs

- The killing and releasing of the resources must happen in a realistic time
- Information about the usage of the platform and about the inner state of the scheduler must be accessible

3 *BatCiGri*

In this Section, we present a solution based on *Batsim* [4] to simulate *CiGri*: *BatCiGri*.

3.1 Hypotheses

We work under the following hypotheses: (*i*) there is only one cluster in the grid, and (*ii*) the only best-effort jobs come from *CiGri*. Regular users of the cluster cannot submit best-effort jobs.

3.2 *Batsim* in a Nutshell

Batsim is a batch scheduler simulator which allows users to test their scheduling algorithms, *i.e.,* how the jobs are mapped to the resources. *Batsim* relies on *Simgrid* [3] for sound simulation models.

The remaining of this section presents two important concepts of *Batsim*: *platforms* and *workloads*.

Platforms *Batsim* platforms, similarly to *Simgrid* platforms, contain information about the underlying platform of the simulation. It contains the number of hosts, the network topology, the speed of the links, the capacity of the disks, etc.

Workloads Workloads contain information about the jobs that will participate in the simulation. There are two main components: `jobs` and `profiles`. Profiles define the behavior of the jobs, *i.e.,* the underlying simulation to use (delay, parallel tasks, SMPI, etc.), execution times, SMPI trace to replay, etc. In a *Batsim* workload, a job belongs to a profile. Each job must have an identifier, a submission time and a requested number of resources. Listing 1 shows a simple example of *Batsim* workload.

A study of the *CiGri* jobs running on the *Gricad* platform [7] gives a statistical description of the execution times of those jobs. This allows us to use a delay model to represent the execution times.

3.3 Two Schedulers

CiGri requires two levels of scheduling. The first level is from *CiGri* to *OAR* for best-effort jobs, and then from *OAR* to the nodes for normal users. Our simulation needs to capture these two levels.

To do so we will have two *Batsim* schedulers: one for the *CiGri* jobs and one for the priority jobs. Each scheduler will manage their own workload but will schedule on the same platform.

As best-effort jobs need to have less priority on the normal jobs, we need a way to kill them. The *CiGri* scheduler will thus only see the free resources of the cluster to perform its schedule of best-effort jobs. On the other hand, the priority scheduler do not see the resources taken by *CiGri* jobs as occupied, and can decide to schedule jobs on those resources. In this case, the *CiGri* scheduler must manage the killing of its jobs.

To be as close to reality, we used the same scheduling algorithms as the real system: conservative backfilling for the priority jobs, and First-Come-First-Served (FCFS) for the *CiGri* jobs.

3.4 Broker

Batsim can only communicate with a single scheduler. However, as seen in the previous section, we have two different schedulers. To deal with this limitation, we used the work done in [13] which implements a message broker between *Batsim* and the schedulers.

The two schedulers connect to the broker and the broker connects to *Batsim*. It filters and redirect the message between the different actors. The main of the work is to manage adaptation of the available resources for the *CiGri* scheduler. When a priority job is submitted, *Batsim* sends a `JOB_SUBMITTED` message to the broker. The broker will then forward this message to the priority job scheduler. If the allocation of resources returned by the scheduler contains best-effort jobs, the broker will inform the *CiGri* scheduler by sending a `REMOVE_RESOURCES` message.

In this case, the *CiGri* scheduler must take care of the killing of the concerned jobs and their resubmission in its

```
{
    "jobs": [
        {
            "id": 1,
            "profile": "cigri",
            "res": 1,
            "subtime": 0
        },
        {
            "id": 2,
            "profile": "cigri",
            "res": 1,
            "subtime": 0
        },
        {
            "id": 3,
            "profile": "cigri",
            "res": 1,
            "subtime": 0
        }
    ],
    "nb_res": 32,
    "profiles": {
        "cigri": {
            "delay": 235.0,
            "type": "delay"
        }
    }
}
```

Listing 1: Example of *Batsim* workload with 3 jobs belonging to the `cigri` profile. Each job requests one resource and are submitted at the start of the simulation.

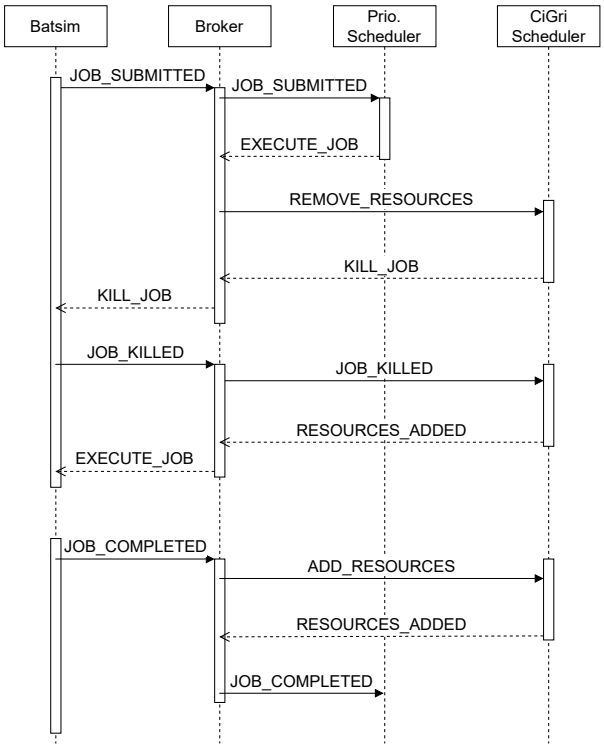


Figure 2: Sequence Diagram representing the killing of best-effort jobs when a new priority job is submitted, as well as when a priority job finishes making its resources idle and thus exploitable by *CiGri*.

queue. When a priority job terminates, its resources become free and thus available to the *CiGri* scheduler. Then, the broker will send a `ADD_RESOURCES` message to *CiGri* to indicate the availability of new resources.

Figure 2 depicts the sequence diagram of a killing of a best-effort job due to a submission of a normal job.

3.5 The *CiGri* Submission Loop

By taking advantage of the `CALL_ME_LATER` event of *Batsim*, we are able to simulate the cyclic behavior of *CiGri*. At every cycle, the *CiGri* scheduler will read the value of the sensors, compute the control error, compute the number of jobs to submit and submit them.

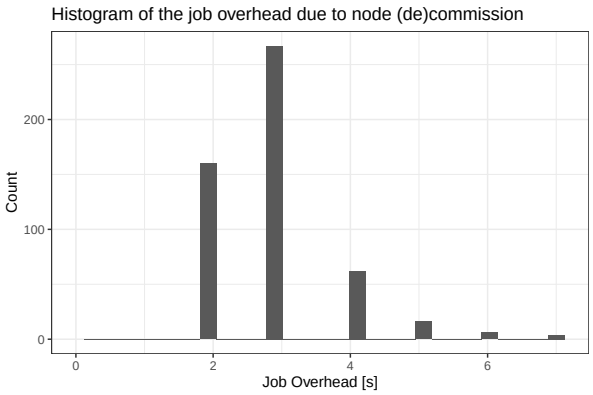
In our case, the sensor is the number of best-effort resources in waiting queue and the number of resources used on the platform. The length of the waiting queue is internal information for the scheduler, whereas the number of resources used is computed indirectly. Remember that the *CiGri* scheduler only sees the resources that are not used by the priority scheduler. Thus, the number of resources currently used on the cluster is the total number of resources minus the number of resources visible by *CiGri* and plus the number of resources used by *CiGri* jobs.

The remaining of the *CiGri* cycle is relatively straightforward and is shown in Listing 2. All the *CiGri* jobs are available at the start of the simulation. This means that in the *Batsim* workload, they are submitted at time 0.

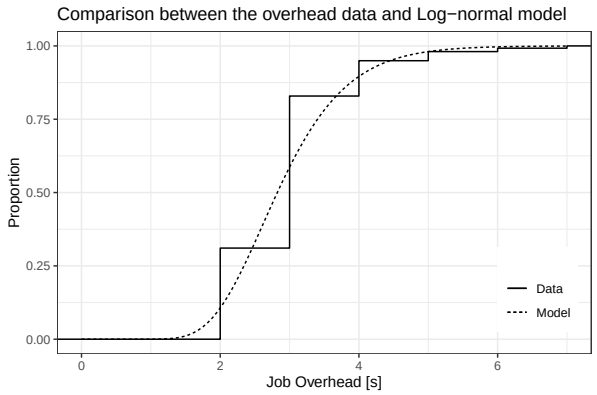
3.6 Workload Adjustments

The synchronization between the real experiments and the simulation is complex, and thus the simulation workload needs to be adjusted to match the real workload.

Starting Delay of *OAR* Performed experiments showed that *OAR* needs about 1 minutes and 30 seconds to start the first jobs after the first submission. This delay should be taken into account in the simulation. From the point of view of the *CiGri* scheduler, this delay can be approximated by not starting the jobs submitting from the first 3 *CiGri* cycles.



(a) Histogram of the distribution of job overhead due to the commission and decommission of resources by *OAR*. Most of the overhead is around 2 and 3 seconds.



(b) Comparison between the empirical cumulative distribution function (CDF) of the overhead (solid) and the CDF of the Lognormal model identified (dashed).

Figure 3: Distribution of the job overheads due to *OAR* commissioning and decommissioning the nodes of the cluster. Figure 3b shows the comparison between the data and the identified model.

```
1  def onRequestedCall(self):
2    # Controller Part ----------------------------------------------------------------------
3    occupied_resources = self.nb_total_resources - len(self.free_resources)
4    sensor = len(self.waiting_queue) + occupied_resources
5
6    self.controller.update_error(sensor)
7    self.controller.update_input()
8    nb_resources_to_submit = self.controller.get_input()
9    # --------------------------------------------------------------------------------------
10
11   # Submission Part ----------------------------------------------------------------------
12   self.add_to_waiting_queue(nb_resources_to_submit)
13   to_schedule_jobs = self.to_schedule_jobs()
14   # --------------------------------------------------------------------------------------
15
16   if len(to_schedule_jobs) > 0:
17       # Ask Batsim to notify for the next cycle
18       self.bs.wake_me_up_at(self.bs.time() + self.cigri_period)
19   else:
20       self.bs.notify_registration_finished()
```

Listing 2: Implementation of the *CiGri* submission loop in *Batsim*. It is triggered by the `CALL_ME_LATER` event. At the end of each loop, we ask *Batsim* to notify us for the next loop (line 18).

Commission and Decommission Times Another source of divergence between simulation and real execution, is the commission and decommission of the resources by *OAR*. This (de)commission time is required to set up the computing nodes for the starting jobs, and to clean the nodes after the termination of the jobs. This delay is not present in *Batsim* and must be considered for realism. We evaluated the (de)commission overhead by submitting jobs which perform an identical and precise amount of work, and compare it to the execution time given by *OAR* (*i.e.,* termination time minus starting time). Figure 3a shows the distribution of overheads in seconds. This distribution shows that the overheads are mostly about 2 or 3 seconds and that the distributions has a long tail. We performed a fitting of a Log-Normal law on the overheads' data to retrieve a statistical model. The fitting yielded that the overheads follow a distribution $Lognormal(1.04, 0.27)$. Figure 3b shows the cumulative distribution functions of the overhead (solid line) and the model (dashed line). This model allows us to generate *Batsim* workloads containing this overhead in the execution time of the jobs.

Killing of Best-Effort Jobs In *Batsim*, when priority jobs are submitted, and they can be scheduled by killing best-effort jobs, the best-effort jobs are immediately stop, and the priority jobs started instantaneously. In practice, the priority jobs spend some time in the waiting queue while the best-effort jobs are being killed and the nodes

cleaned and set up. This delay can be taking into account in the description of the priority jobs. The execution time in *Batsim* must also contain this delay.

4 Evaluation

In this Section, we evaluate the quality of the simulation.

4.1 Experimental Protocol

For both the real system and the simulated one we will conduct the same scenario. There are 500 *CiGri* jobs with an execution time of 235 seconds. The submission loop of *CiGri* is called every 30 seconds in order to see how the system respond to delay in the control input. After 2000 seconds, a priority job is submitted and takes half of the resources of the cluster for 1800 seconds. The controller of *CiGri* aims to regulate the quantity $w_k + r_k$ around the value 64 (which is the double of the number of resources in the cluster).

4.2 Experimental Setup

The real experiments were carried on the `dahu` cluster of *Grid'5000* [1] where the nodes have 2 Intel Xeon Gold 613 with 16 cores per CPU and 192 GiB of memory. The reproducibility of the deployed environment is ensured by *NixOS Compose* [10].

We deploy 3 nodes: one for the *OAR* server, one for *CiGri*, and one for the *OAR* cluster. We do not deploy 32 nodes for the cluster, but instead deploy a single node and define 32 OAR resources.

4.3 Execution time

One of the motivation of this study if the cost in time in resources of experiments. Real experiments require deploying 3 resources (around 10 minutes), and then to execute the scenario (around 1h20 minutes). In total, a single execution of the scenario consumes around 9 CPU hours.

In comparison, a simulation requires a single CPU, and needs 2 seconds to complete, thus consuming approximately 5.5×10^{-4} CPU hours

4.4 Signals Comparison

For the simulation of *CiGri* to be useful, we need the signals of interest to have the same properties and behave the same in both simulation and real experiments. The signals of interest are:

- the number of best-effort resources in the waiting queue
- the number of currently used resources on the cluster
- the dynamic of a *CiGri* submission (*i.e.,* the time it takes to see the impact of a submission)

Figure 4 shows the comparison of the signals of interested between experiments of the same scenario executed in simulation (red) and deployed on real machines (blue). The signals appear to be in sync. The amplitude do differ, as can be observed around 500 seconds. The real system is obviously more sensible to noise. This noise can be noticed when looking at the used resources (top left graph on Figure 4). The cluster in the simulation is always full, whereas the cluster during real experiments is not (*e.g.,* at 1000, 2000, 4500 seconds).

4.5 Gantt Charts Comparison

Figure 5 compares the resulting Gantt charts of the experiment for the simulation (top) and real execution (bottom). We notice that there are "gaps" in the real schedule (*e.g.,* at time 1500 seconds on resource 24). These gaps create a lag in the schedule which also impact the signals.

This lag comes from *OAR* scheduling algorithm. Once the *OAR* decided to start to compute a scheduler, if any job arrives during the execution of the scheduler, those jobs will not be taken into account until the next schedule call. Taking into account this lag in the simulation is complex, as *Batsim* is responsible for the management of the simulation time, and because the time "stops" during the computation of the schedule.

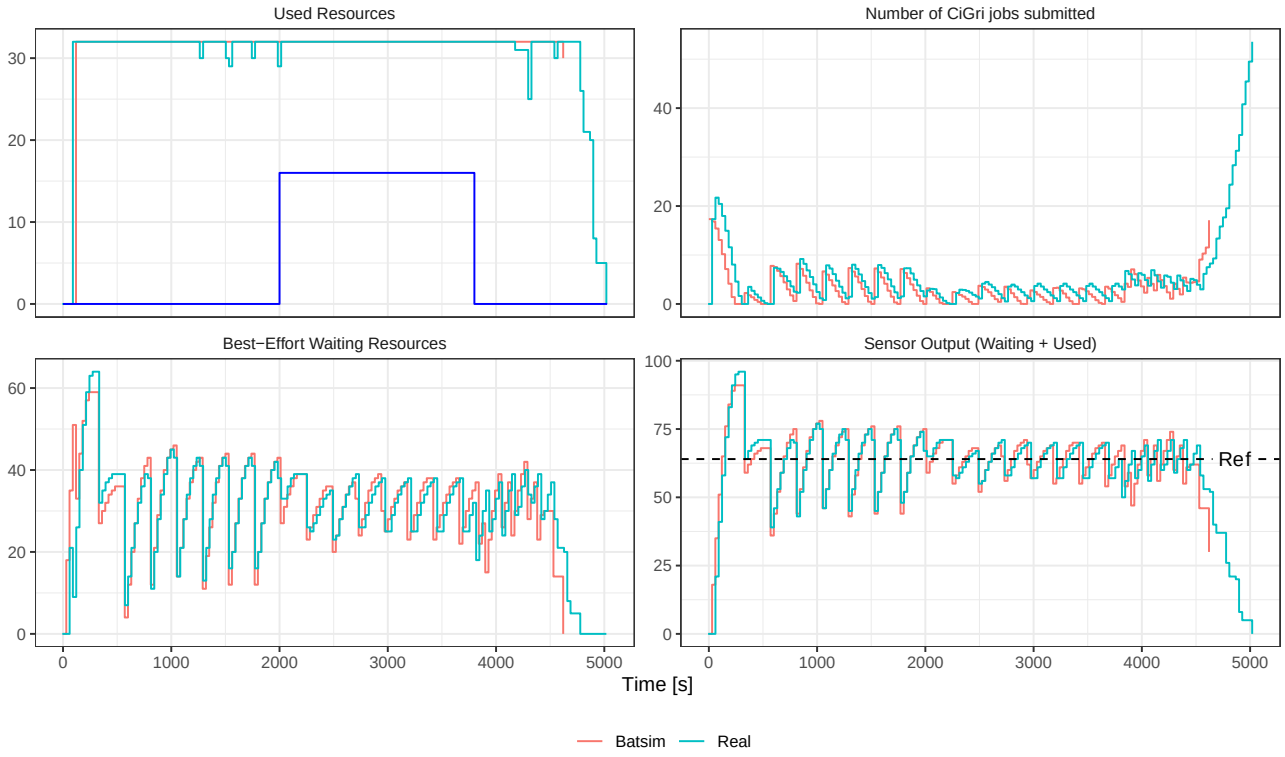


Figure 4: Comparison of the signals of interest for the same experiment executed in simulation with *Batsim* (red) and deploy (blue). Signals appear to be in sync, but some amplitudes might differ.

5 Conclusion

Distributed experiments are complex and costly. Simulation techniques can help reduce the cost of such experiments. However, simulators rely on models that can lose information compared to the real system. In this paper, we implemented the essential behavior of *CiGri*, a grid middleware, in *Batsim*. Real experiments with *CiGri* requires 3 compute nodes for several hours, while simulation last a few seconds on a laptop. We compared the behavior and similarities of signals of interest of our system in simulation and real experiments. We had to modify the workload of the simulation to match the different overheads induced by the real system. Results showed satisfying quality of signals in simulation.

In this paper, we only focused on the execution time part of the jobs. However, in our *CiGri* works [8, 9] we control the submission of *CiGri* jobs to *OAR* in order regulate the load of a distributed file-system. Taking into account a parallel file-system in *Batsim* is feasible [13]. However, the current limitation of *Batsim* is the lack of probe mechanism to sense internal information and states.

References

[1] Daniel Balouek et al. "Adding Virtualization Capabilities to the Grid'5000 Testbed". In: *Cloud Computing and Services Science*. Ed. by Ivan I. Ivanov et al. Vol. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, pp. 3–20. ISBN: 978-3-319-04518-4. DOI: 10.1007/978-3-319-04519-1_1.

[2] N. Capit et al. "A batch scheduler with high level components". en. In: *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.* Cardiff, Wales, UK: IEEE, 2005, 776–783 Vol. 2. ISBN: 978-0-7803-9074-4. DOI: 10.1109/CCGRID.2005.1558641. URL: http://ieeexplore.ieee.org/document/1558641/ (visited on 05/25/2020).

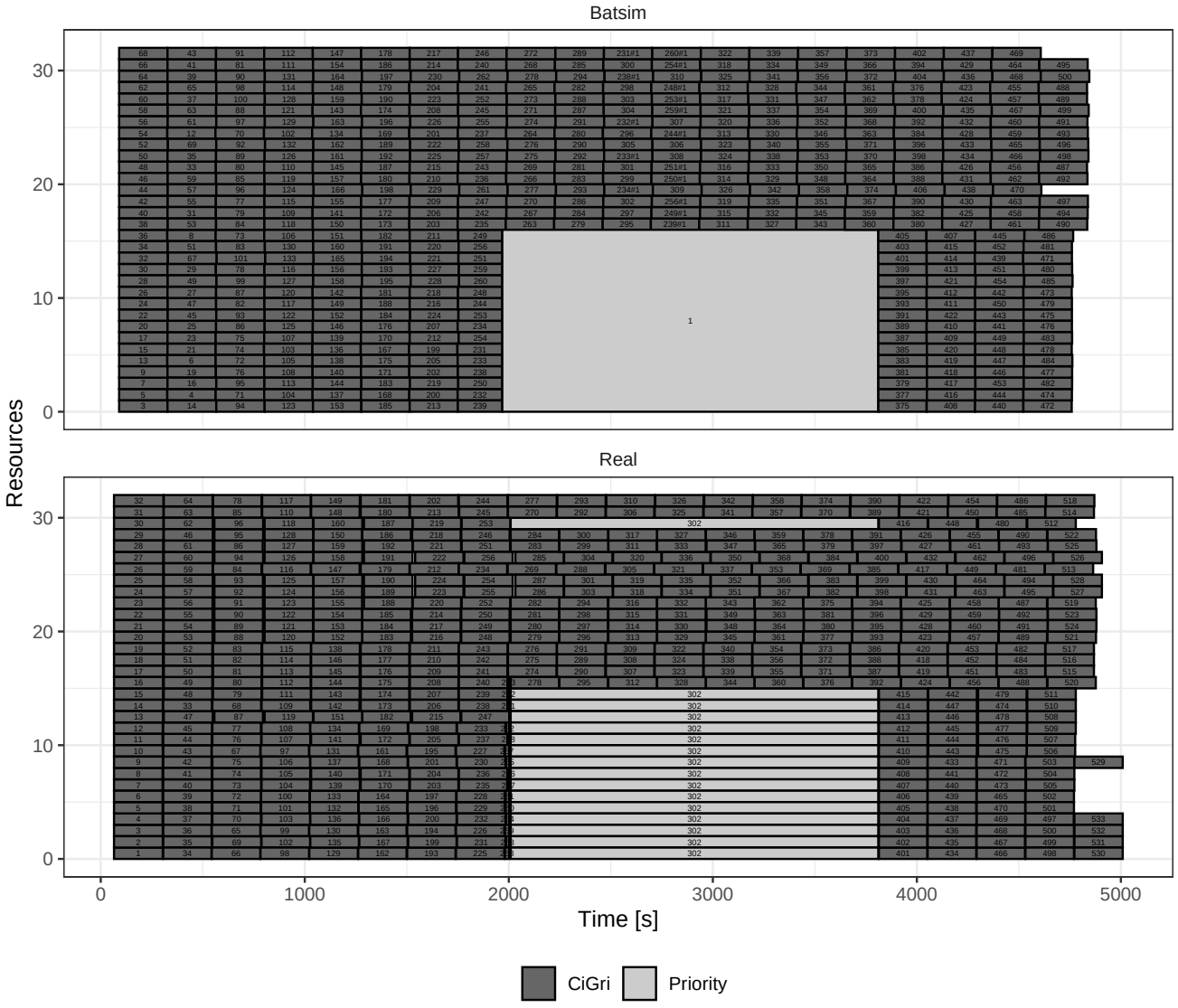


Figure 5: Comparison of the Gantt charts for the simulation (top) and real experiment (bottom) of the same scenario. We observe a small lag, which is due to *OAR*, but both schedules are similar.

[3] Henri Casanova et al. "Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms". In: *Journal of Parallel and Distributed Computing* 74.10 (June 2014), pp. 2899–2917. URL: http://hal.inria.fr/hal-01017319.

[4] Pierre-François Dutot et al. "Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator". In: *20th Workshop on Job Scheduling Strategies for Parallel Processing*. Chicago, United States, May 2016. URL: https://hal.archives-ouvertes.fr/hal-01333471.

[5] Cristian Galleguillos, Zeynep Kiziltan, and Alessio Netti. "Accasim: an HPC simulator for workload management". In: *High Performance Computing: 4th Latin American Conference, CARLA 2017, Buenos Aires, Argentina, and Colonia del Sacramento, Uruguay, September 20-22, 2017, Revised Selected Papers 4*. Springer. 2018, pp. 169–184.

[6] Yiannis Georgiou, Olivier Richard, and Nicolas Capit. "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.

[7] Quentin Guilloteau, Olivier Richard, and Éric Rutten. "Étude des applications Bag-of-Tasks du méso-centre Gricad". In: *COMPAS 2022 - Conférence francophone d'informatique en Parallélisme, Architecture et Système*. Amiens, France, July 2022, pp. 1–7. URL: https://hal.archives-ouvertes.fr/hal-03702246.

[8] Quentin Guilloteau et al. "Controlling the Injection of Best-Effort Tasks to Harvest Idle Computing Grid Resources". In: *ICSTCC 2021 - 25th International Conference on System Theory, Control and Computing*. Iași, Romania, Oct. 2021, pp. 1–6. DOI: 10.1109/ICSTCC52150.2021.9607292. URL: https://hal.inria.fr/hal-03363709.

[9] Quentin Guilloteau et al. "Model-free control for resource harvesting in computing grids". In: *Conference on Control Technology and Applications, CCTA 2022*. Trieste, Italy: IEEE, Aug. 2022. URL: https://hal.archives-ouvertes.fr/hal-03663273.

[10] Quentin 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. Heidelberg, Germany, Sept. 2022, pp. 1–12. URL: https://hal.science/hal-03723771.

[11] Jeffrey O Kephart and David M Chess. "The vision of autonomic computing". In: *Computer* 36.1 (2003), pp. 41–50.

[12] Dalibor Klusáček, Mehmet Soysal, and Frédéric Suter. "Alea–complex job scheduling simulator". In: *Parallel Processing and Applied Mathematics: 13th International Conference, PPAM 2019, Bialystok, Poland, September 8–11, 2019, Revised Selected Papers, Part II 13*. Springer. 2020, pp. 217–229.

[13] Michael Mercier. "Contribution to High Performance Computing and Big Data Infrastructure Convergence". en. PhD Thesis. Universite Grenoble Alpes, 2019.