
Master of Science in Informatics at Grenoble
Master Informatique
Specialization MoSIG

# Simulation of a Kubernetes Cluster with Validation in Real Conditions

**LARUE Théo**

Defense Date, 2020

Research project performed at Laboratoire d'Informatique de Grenoble

Under the supervision of:
Michael Mercier

Defended before a jury composed of:
Head of the jury
Jury member 1
Jury member 2

September 2020

**Abstract**

TODO : rework this with the new intro

The rise of containerized applications has provided web platforms with much more control over their resources than they had before with their physical servers. Soon enough, developers realized they could go even further by automating container management operations to allow for even more scalability. The Cloud Native Computing Foundation was founded in this context, and developed Kubernetes which is a piece of software capable of container orchestration, or in other words, container management. Now, as we observe a convergence between HPC (High Performance Computing) and the Big Data field where Kubernetes is already the standard for some applications such as Machine Learning, discussions about leveraging containers for HPC applications rose and interest in Kubernetes has grown in the HPC community. One of the many challenges the HPC world has to face is scheduling, which is the act of allocating tasks submitted by users on available resources. In order to properly evaluate and develop schedulers researchers have used simulators for decades to avoid running experiments in real conditions, which is costly both in time and resources. However, such simulators do not exist for Kubernetes or are not open to the public. While the default scheduler works great for most of the Cloud Native infrastructures Kubernetes was designed for, some teams of researchers would rather be able to experiment with different batch processing policies on Kubernetes as they do with traditional HPC. Our goal in this master thesis is to describe how we developed Batkube, which it is an interface between Kubernetes schedulers and Batsim, a general purpose infrastructure simulator based on the Simgrid framework and developed at the LIG.

**Acknowledgement**

I would like to express my sincere gratitude to .. for his invaluable assistance and comments in reviewing this report... Good luck :)

**Résumé**

Abstract mais en franchais

# Contents

**Abstract** i

**Acknowledgement** i

**Résumé** i

**1 Introduction** 1

**2 State of the art** **3**
2.1 Context: Cloud Native Computing . . . 3
2.2 Studying computer infrastructures . . . 3
*in vivo* and *in vitro* studies . . . 4
*in silico* studies, or simulation . . . 4
Kubernetes simulation . . . 4
The scheduling problem . . . 5
2.3 Batsim concepts and objectives . . . 5
SimGrid . . . 5
Related projects . . . 5

**3 Implementation** **7**
3.1 Batsim . . . 7
3.2 Kubernetes concepts . . . 7
3.3 Objectives . . . 8
3.4 Technical challenges . . . 9
3.4.1 Translation . . . 9
3.4.2 Time hijack . . . 9
3.4.3 Time synchronisation . . . 10
3.4.4 Re-building the API . . . 10

**4 Evaluation** **11**

**5 Conclusion** **13**

**Bibliography** **15**

# — 1 —
# Introduction

The need for scalable computing infrastructure has increased tremendously in the last decades. Nearly every field of computer science, from research to the service industry, now needs a proper infrastructure and by 2025, computation technology could reach a fourth of the global electricity spending[1]. Even the public sector is now in need for efficient distributed infrastructure as the concept of smart cities is developing.

Organizations generally know what type of infrastructure will meet their needs. It can take the form of Big Data centers to store and analyze data, High-Performance Computers for computing intensive tasks or GPU banks for machine learning or crypto-currency mining. However, studying those infrastructures extensively is much more challenging. As these computers reach scales in the order of warehouses[2], quantifying a system's performance under varying loads, applications, scheduling policies and system size quickly becomes undoable without expensive real world experiments. In fact, the nature of scheduling problems[7] alone make theoretical studies hard. This is an issue for organizations as they rely on thoses studies to determine the size of the required system or choose optimal scheduling policies.

Simulation allows to tackle these issues by enabling users to draw conclusions empirically without the need to fire up real workloads. Indeed, running an entire experimental campaign on a real system represents consequencial costs both in time and money. With simulation, The gain in both time and spent energy can be extreme : a HPC job spanning months on a real system can be resolved in a matter of minutes on any domestic computer. Another major point is that it also brings reproducibility to these experiments, that otherwise would have to be run on the exact same systems as their first iteration. With simulation, one can recreate the same conditions for any experiment anywhere they want, and expect the same results.

However, simulations need to be run with sound models for the results to be exploitable and in that regard, simulators usually fall under several pitfalls[8]. Very often simulators are implemented at the same time as new schedulers or Resource and Jobs Management Systems[1] in order to validate their algorithms. Thus, they are strongly coupled together and are not usable with any other software. They are either shipped with the software itself or worst, they are never released and discarded at the end of the development process. Moreover, still according to [8], strong coupling may lead to unrealistic models. In that case cluster resources can be accessed with ease by the scheduler, resulting in it having very precise information about the system state to take its decisions. This conflicts with the real world as a scheduler may not have access to all the information it wants, or may suffer from latency when getting it from the system.

To try and assess these issues a team of researchers at the LIG developed Batsim[4] which is a general purpose infrastructure simulator with modularity and separation of concerns in mind. Batsim is based on SimGrid[3] which is a framework for developing simulators for distributed computer systems. Simgrid is now a 20 years old framework that has been used in many projects[2], making it a sound choice to run

[1]The RJMS is the software at the core of the cluster. It is a synonym for a scheduler and manages resources, energy consumption, users' jobs life-cycle and implements scheduling policies.

[2]https://simgrid.org/usages.html

scalable and accurate models of the reality.

Batsim was designed to support algorithms written in any languages, as long as they support its communication protocol. It means that, while any scheduler found in the wild can potentially be run on a Batsim simulation, they still have to be adapted to make them compatible. This master's project is dedicated on developing an interface between Batsim and Kubernetes[3] schedulers in order to run Kubernetes clusters simulations. Kube[4] is an open source container management software widely exploited in the industry for its ease of use and wide range of capabilities. It has freed developers from the cumbersome task of setting up low level software infrastructure on their servers and automates maintenance, scaling and administration of their applications. For all these reasons it has become a de-facto solution for any organization that wishes to build new internet platforms from the ground up.

TODO : what we where able to do (summary of the simulator capabilities, experimentations, results)

[3] https://github.com/kubernetes/kubernetes/

[4] Another term to designate Kubernetes. It is also sometimes called k8s.

# — 2 —
# State of the art

## 2.1 Context: Cloud Native Computing

In the early stages of application development, organizations used to run their services on physical servers. With this direct approach came many challenges that needed to be coped with manually like resources allocation, maintainability or scalability. In an attempt to automate this process developers started using virtual machines which enabled them to run their services regardless of physical infrastructure while having a better control over resources allocation. This led to the concept of containers which takes the idea of encapsulated applications further.

Containers can be thought of as lightweight virtual machines. Unlike the latter, containers share the same kernel with the host machine but still allow for a very controlled environment to run applications. There are many benefits to this : separating the development from deployment, portability, easy resource allocation, breaking large services into smaller micro-services or support of continuous integration tools (containers greatly facilitate integration tests).

The CNCF[1] (Cloud Native Computing Foundation) was founded in the intent of leveraging the container technology for an overall better web. In a general way, we now speak of these containerized and modular applications as cloud native computing :

*"Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.*

*These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil."*[2]

Kubernetes[3] is the implementation of this general idea and was anounced at the same time as the CNCF. It aims at automating of the process of deploying, maintaining and scaling containerized applications. It is industry grade and is now the de-facto solution for container orchestration.

## 2.2 Studying computer infrastructures

Studying an entire computing infrastructure is not an easy feat, first because every infrastructure is unique. There are as many types of infrastructure as there are use cases, each having a different vision on efficiency and what metrics are critical to the system: latency, bandwith, resource availability, computational power or cost effectiveness (the latter boils down to energy efficiency). This variety of purposes translates to the type of hardware used and the topology the infrastructure. Some systems are centralized like HPC and Data

[1] `https://www.cncf.io/`

[2] `https://github.com/cncf/toc/blob/master/DEFINITION.md`

[3] `https://kubernetes.io/`

Centers, others are meant to be used from a distance like Cloud Computing infrastructures and others are decentralized like Grid Computingn Volunteer Computing and Peer to Peer computing. There are as many systems as there are objectives to be achieved.

As a consequence, there are no general tools to study those systems. Furthermore, as the biggest supercomputers are approaching the exascale barrier[4] and consist of thousands of nodes with millions of cpu cores (more than 7M for the new "Fugaku" Japanese supercomputer), no human would be capable of building a general mathematical model that would be accurate enough to predict the behavior of those systems under varying conditions. Also, interactions between the various components of those systems may lead to unexpected behavior[5] that can hardly be predicted. Then, in order to extensively experiment on a given system, there are 3 options left as described in [6]: *in vivo*, *in vitro* and *in silico* studies, which correspond respectively to experiments on real testbeds, emulation and simulation.

The next sections build upon [6] and [3] by the team behind the SimGrid[3] simulator, and are aimed at depicting the current landscape of experimentation on distributed systems.

### *in vivo* and *in vitro* studies

The most direct approach to study an infrastructure is running *in vivo* experiments, that is to say running experiments on a real testbed. This will produce the most accurate results, but one can already guess this approach is not viable for extensive studies on the bigger systems.

TODO

### *in silico* studies, or simulation

Why: explained earlier.

Problems with simulation: often unrealeased simulators, or designed for a specific project, or short lived (OptorSim) -> This is why Batsim was created.

Simulators specific to platforms: YARNSim, SLURM simulator[5] Exemples of papers with custom unreleased simulators: [10] by the same guys who made kubernetes-simulator.

Related projects list (will definetly not include everything. This is not aimed at being a report on the entire simulation world).

- SimGrid, GridSim, CloudSim, GroudSim (to cite the most important).
- Other simulators in unrelated domains: SimBA (volunteer computing), PeerSim, OverSim (peer to peer), WRENCH (workflows).
- HPC simulation: off-line vs on-line
- Interconnected networks Simulation: INSEE (environment for interconnected networks), SICOSYS. Aimed to be used with other tools like SIMICS to extend th ecapabilities.
- Low level simulation: SIMICS, RSIM and SimOS (multiprocessor systems).
- discontinued/old projects: GSSIM, Simbatch

### Kubernetes simulation

Kubernetes simulation: k8-cluster-simulator, joySim. FOCUS ON THAT

[4] https://www.top500.org/news/japan-captures-top500-crown-arm-powered-supercomputer/

[5] https://github.com/ubccr-slurm-simulator/slurm_simulator

### The scheduling problem

TODO: What to do with this section?

In particular, this work is targeted at experimenting with scheduling in a distributed system driven by Kubernetes.

> **schedule** *n.* : A plan for performing work or achieving an objective, specifying the order and allotted time for each part.

In a general way, scheduling is the concept of allocating available resources to a set of tasks, organizing them in time and space (the resource space). The resources can be of any nature, and the tasks independent from each others or linked together.

In computing the definition remains the same, but with automation in mind. Schedulers are algorithms that take as an input either a pre-defined workload, which is a set of jobs to be executed - the tasks are called jobs in this context -, or simple jobs submitted over time by users in an unpredictable manner. In the latter case, the jobs are added to a queue managed by the scheduler. Scheduling is also called batch scheduling or batch processing, as schedulers allocate batches of jobs at a time. Jobs are allocated on machines, virtual or physical, with the intent of minimizing the total execution time, equally distributing resources, minimizing wait time for the user or reducing energy costs. As these objectives often contradict themselves, schedulers have to implement compromises or focus on what the user really needs or requires.

The scheduler has many factors to keep in mind while trying to be as efficient as possible, such as :

- Resource availability and jobs resource requirements
- Link between jobs (some are executed in parallel and need synchronization, some are independent)
- Latency between compute resources
- Compute resources failures
- Jobs priority
- Machine shutdowns and restarts
- Data locality

All these elements make scheduling a very intricate problem that is at best polynomial in complexity, and often NP-hard ([9], [7]).

## 2.3 Batsim concepts and objectives

TODO General concepts of batsim: Language agnostic, ease of use, aimed at experimenting on scheduling policies, deterministic (reproducibility is at the core of batsim), coarse grain simulation.

### SimGrid

TODO : Quick description of SimGrid objectives and general concepts. (General purpose, large scale simulation).

Does this justify a whole subsection? Isn't a paragraph enough for this? I don't need to dig deep into SimGrid inner mechanics as this work is at the higher level.

### Related projects

Aléa and Accasim

# — 3 —
# Implementation

## 3.1 Batsim

## 3.2 Kubernetes concepts

The basic processing unit of Kubernetes is called a **pod** which is composed of one or several containers and volumes[1]. In the cloud native context a pod most often hosts a service or micro-service.

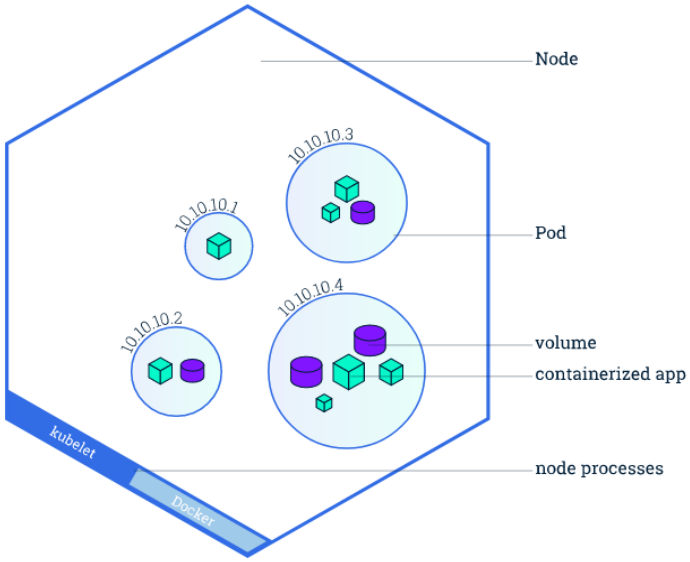


Figure 3.1: Node overview
**Source:** *https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/*

Pods are bundled together in **nodes** (figure 3.1) which are either physical or virtual machines. They represent another barrier to pass through to access the outside world which can be useful to add layers of security or facilitate communication between pods. Nodes take the idea of containerisation further by encapsulating the already encapsulated services. Each node runs at least one pod and also one **kubelet** which is a process responsible for communicating with the rest of Kubernetes (or more precisely, with the master node which in turns communicates with the api server). A set of nodes is called a **cluster**. Each Kubernetes instance is responsible for running a cluster.

[1] A volume is some storage space on the host machine that can be linked to containers, so they can read persistent information or store data in the long term

Kubernetes revolves its API server which is its central component (figure 3.2). The majority of operations between components go through this REST API like user interactions through kubectl or scheduling operations.

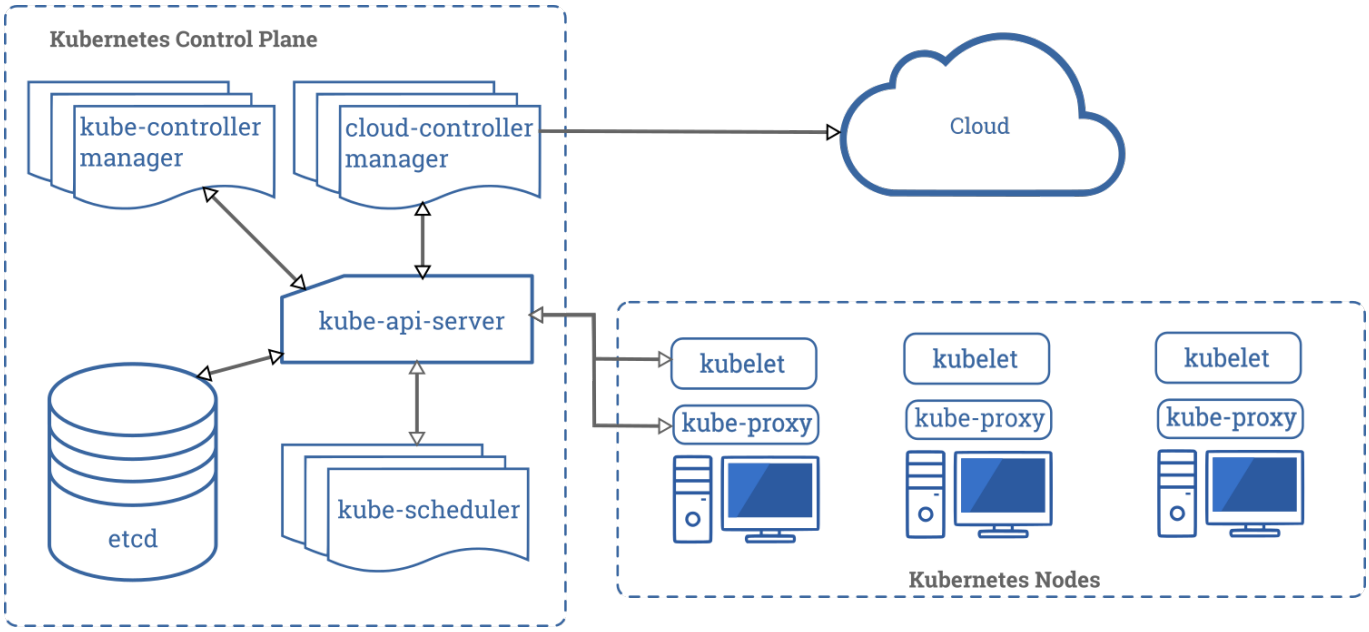


Figure 3.2: Components of Kubernetes
**Source:** *https://kubernetes.io/docs/concepts/overview/components/*

## 3.3 Objectives

The goal of this project is to design and implement Batkube, which will be an interface between Batsim and Kubernetes schedulers. With this interface, we want to compare Batsim results gainst data from a real Kubernetes cluster, given HPC workloads.

## 3.4 Technical challenges

### 3.4.1 Translation

### 3.4.2 Time hijack

TODO

**Algorithm 1:** Requester loop

```
   Input: req: request channel, res: result channel map
1  while Batkube is not ready do
2      wait
3  requests = []request
4  while req is not empty do
5      m = <- req /* Non blocking receive                                  */
6      requests = append(requests, m)
7  sendToBatkube(requests) /* Only requests with duration > 0 are actually sent.
       Batkube will always anwser.                                         */
8  now = responseFromBatkube()
9  for m in range requests do
10     res[m.id] <-now /* The caller continues execution upon reception    */
```

**Algorithm 2:** Time request (time.now())

```
   Result: Current simulation time
   Input: d: timer duration, req: request channel, res: response channel map
   Output: now : simulation time
1  if requester loop is not running then
2      go runRequesterLoop() /* There can on ly be one loop runing at a time */
3  id = newUUID()
4  m = newRequestMessage(d, id) /* Requests are identified using uuids     */
5  resChannel = newChannel()
6  res[id] = resChannel /* A channel is associated with each request       */
7  req <- m /* The code blocks here until request is handled               */
8  now = <-resChannel /* The code blocks here until response is sent by the
       requester loop                                                      */
9  return now
```

### 3.4.3 Time synchronisation

**Algorithm 3:** Synchronisation with timers

```
Input: timeout: time.Duration; simulationTimestep: time.Duration
stopWaitingForMessages := false
while !stopWaitingForMessages do
    getTimerRequests() /* Translate timers requested by the scheduler to
        CALL_ME_LATER events                                              */
    getSchedulerDecisions() /* Translate scheduler decisions to Batsim messages */
    if decisionReceived() or timeSinceLastMessage() > timeout then
        stopWaitingForMessages = true
/* We keep track of all REQUESTED_CALL events yet to come                */
if durationToNextRequestedCall > simulationTimestep then
    addCallMeLater() /* Add a CALL_ME_LATER event with timestamp now +
        simulationTimestep                                                */
```

**Algorithm 4:** Synchronisation with incremental time increases

```
Input: simulationTimeout: time.Duration; incrementTimeStep: time.Duration; incrementValue:
       time.Duration
stopWaitingForMessages := false
incremented := 0
while !stopWaitingForMessages do
    getSchedulerDecisions()
    if decisionReceived() or incremented >= simulationTimeout then
        stopWaitingForMessages = true
    else if timeSinceLastIncrement() > incrementTimeStep then
        /* this last condition is here to slow down this process and give time
           for the scheduler to take its decisions                        */
        now = now + incrementValue
        incremented = incremented + incrementValue
addCallMeLater() /* Add a CALL_ME_LATER event with timestamp now +
    incrementValue                                                        */
```

### 3.4.4 Re-building the API

# — 4 —
# Evaluation

# — 5 —
# Conclusion

# Bibliography


[1] Anders Andrae. "Total consumer power consumption forecast". In: *Nordic Digital Business Summit* 10 (2017).

[2] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. "The datacenter as a computer: Designing warehouse-scale machines". In: *Synthesis Lectures on Computer Architecture* 13.3 (2018), pp. i–189.

[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] Dalibor Klusáček and Šimon Tóth. "On Interactions among Scheduling Policies: Finding Efficient Queue Setup Using High-Resolution Simulations". In: *Euro-Par 2014 Parallel Processing*. Ed. by Fernando Silva, Inês Dutra, and Vítor Santos Costa. Cham: Springer International Publishing, 2014, pp. 138–149. ISBN: 978-3-319-09873-9.

[6] Arnaud Legrand. "Scheduling for large scale distributed computing systems: approaches and performance evaluation issues". PhD thesis. 2015.

[7] Peter Brucker, Sigrid Kunst. *Complexity results for scheduling problems*. June 29, 2009. URL: http://www2.informatik.uni-osnabrueck.de/knust/class/ (visited on 06/10/2020).

[8] Millian Poquet. "Simulation approach for resource management". Theses. Université Grenoble Alpes, Dec. 2017. URL: https://tel.archives-ouvertes.fr/tel-01757245.

[9] J. D. Ullman. "NP-Complete Scheduling Problems". In: *J. Comput. Syst. Sci.* 10.3 (June 1975), 384–393. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(75)80008-0. URL: https://doi.org/10.1016/S0022-0000(75)80008-0.

[10] Hidehito Yabuuchi, Daisuke Taniwaki, and Shingo Omura. *Low-latency job scheduling with preemption for the development of deep learning*. 2019. arXiv: 1902.01613 [cs.DC].