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

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

— 1 —
Introduction

1.1 Kubernetes

1.1.1 Kubernetes overview

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 infrasctucture while having a better control over resources allocation. This led to the concept of containers which takes the idea of encapsulated applications further.

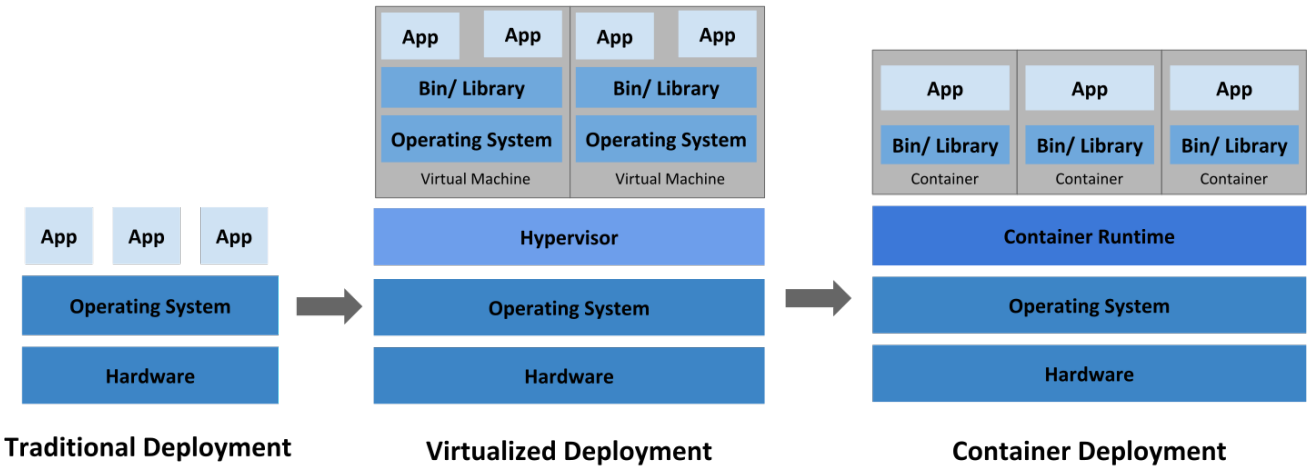

Figure 1.1: Evolution of application deployment.
Source: *https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/*

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

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

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.

Kubernetes concepts

The basic processing unit of Kubernetes is called a **pod** which is composed of one or several containers and volumes[4]. In the cloud native context a pod most often hosts a service or micro-service.

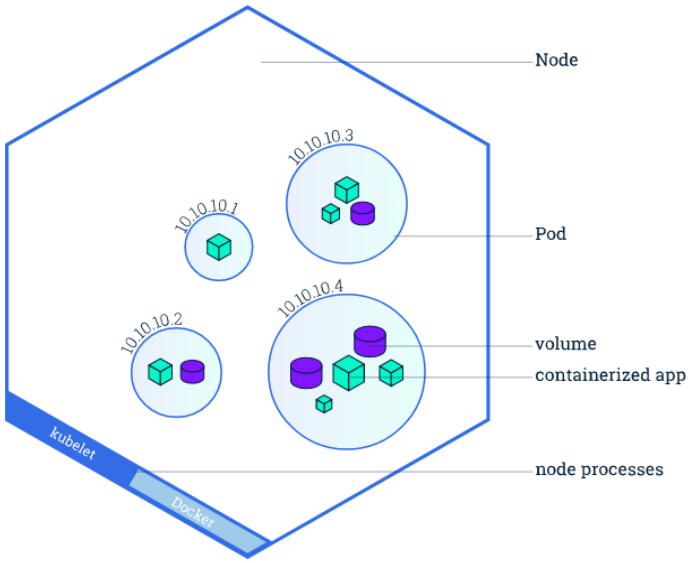

Figure 1.2: Node overview
Source: *https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/*

Pods are bundled together in **nodes** (figure 1.2) 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.

Kubernetes revolves its API server which is its central component (figure 1.3). The majority of operations between components go through this REST API like user interactions through kubectl or scheduling operations.

[2] `https://github.com/cncf/toc/blob/master/DEFINITION.md`
[3] `https://kubernetes.io/`
[4] 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

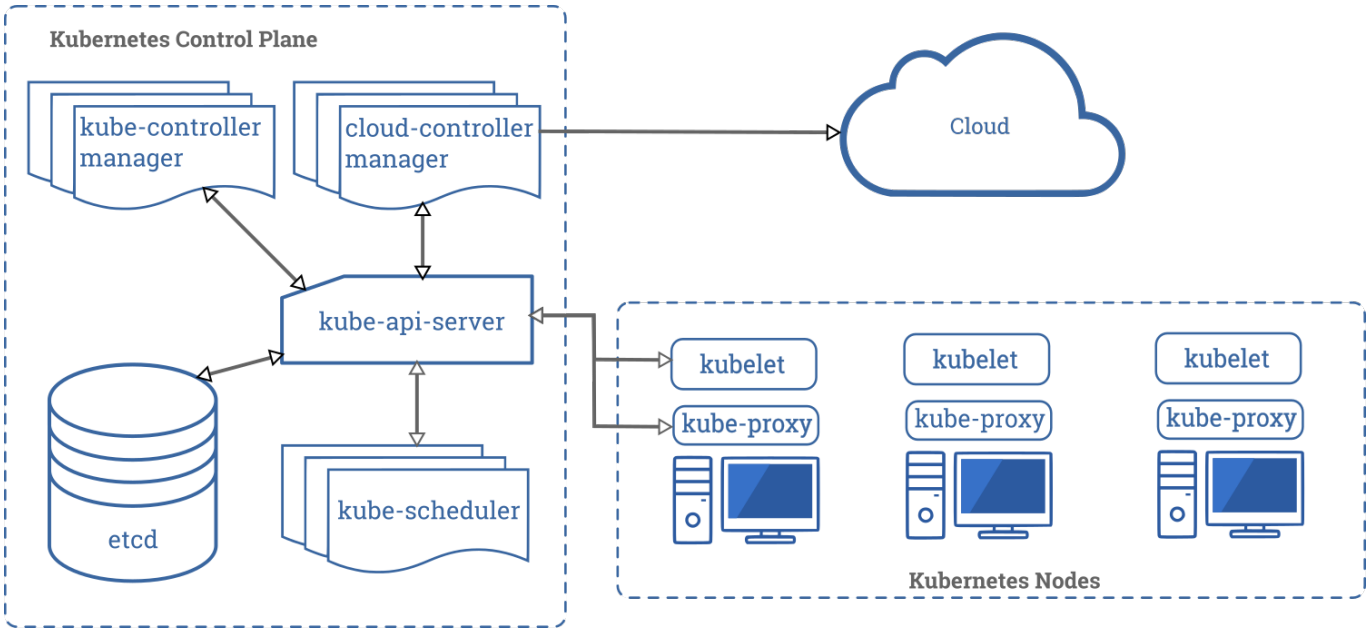

Figure 1.3: Components of Kubernetes
Source: *https://kubernetes.io/docs/concepts/overview/components/*

1.1.2 HPC and Kubernetes

The difference between HPC (High Performance Computing) and Cloud Native computing lies in the workloads they are intended to tackle.

A general definition of HPC would be : "*High Performance Computing (HPC) most generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer in order to solve large problems in science, engineering, or business.* [5]". HPC can either refer to "High Performance Computing" or "High Performance Computer" but it is generally clear which one it refers to, given the context.

A HPC workload is composed of numerous tasks hungry for computational resources which are executed in parallel on different machines (that we can refer to as compute resources or nodes). These tasks may be completely independant like when different users each submit a single task, or they may also be tightly coupled together as when a single user submits a job that is composed of several tasks than can be run in parallel. In that case, the whole system becomes very sensitive to latency as these tasks have got to communicate together. This is done through MPIs (Message Passing Interfaces) which represent a large part of the HPC field.

Kubernetes is now the standard for AI and Machine Learning as shown by the many efforts at making this coupling an efficient environment[3][6][5], which brought an increasing interest for container driven HPC aswell and Kubernetes for HPC in particular. Batch schedulers such as kube-batch[6] have been implemented for kube, and numerous HPC applications like slurm[7] have been containerized aswell.

Indeed, containers have many advantages that HPC users can benefit from. Here are some notable ones:

- First off, research has shown that Kuberenetes offer similar performance to more standard bare metal HPC[1].

- Users will get the same environment everywhere making up for a uniform and standardized workplace.

[5] `https://wwwen.uni.lu/university/high_performance_computing`
[6] `https://github.com/kubernetes-sigs/kube-batch`
[7] `https://slurm.schedmd.com/containers.html`

- Portability : users could seamlessly hop from one infrastructure to another based on their needs and criteria like price, performance, and capabilities rather than compatibility.
- Encapsulation : HPC applications often rely on complex dependencies that can be easily concealed into containers.

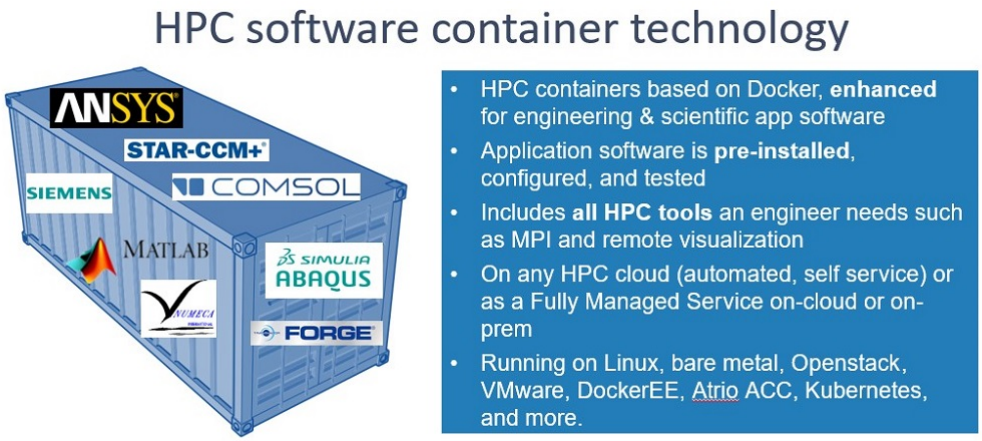

Figure 1.4: The container technology for HPC
Source: *https://www.hpcwire.com/2019/09/19/kubernetes-containers-and-hpc/*

Despite all those advantages, Kubernetes is not ready yet to be used in proper HPC environment because it lacks vital components like a proper batch job queuing system, and support for MPI applications. It cannot yet compete against the very well established HPC ecosystem, however efforts are being made in this direction. This master thesis is focused on a part of this problem : Kubernetes simulation for batch schedulers evaluation.

1.2 Scheduling

> **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 independant 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 independant)
- 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[4]. Moreover, with the growing complexity of modern Resource and Job Management Systems (RJMS)[8] and the wide variety in infrastructures, scheduler performances are rather inpredictable.

1.3 Simulating an infrastructure

Since it is very difficult to establish a model for a scheduler on a certain infrastructure to try and predict is performances on a given situation, evaluating a scheduler often boils down to testing it against real scenarios - that is to say with a real workload, on a real infrastructure. Indeed, that is costly both in time and resources and schedulers can't be evaluated extensively or "on the go" while being developed. One solution to this problem is to run simulations of the infrasctuctures on which they run.

[8]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 —
State of the art

2.1 Batsim

2.2 Infrastructure simulators

Developing a scheduler implies being able to test its performances throughout the development process, however, testing in real conditions is time consuming and expensive. Organizations can either have enough resources to cover these costs, or test their scheduler against a simulation.

Kubernetes cluster simulations is an open problem and is the subject of this master project. Our approach relies on the Batsim[**batsim**] infrastructure simulator, which is itself built upon Simgrid[**simgrid**]. Batsim is currently mostly used to simulate HPC infrastructures but was designed to be able to simulate any kind of infrastructure and therefore is theoretically able to simulate any Kubernetes cluster, moreover, Kubernetes was designed to run services but is capable of handling High-Performance Computing[2]. This project aims at adapting Batsim so it can evaluate Kuberenetes schedulers.

2.3 Kubernetes schedulers

kube-scheduler

Not exactly a batch scheduler, it is the default scheduler for Kubernetes made by the CNCF

kube-batch

Poseidon

bashScheduler by rothgar

random-scheduler by Banzaicloud

k8s-custom-scheduler by IBM

scheduler by kelseyhightower

— 3 —
Problematic

3.1 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.

3.2 Translation

3.3 Synchronization

— 4 —
Implementation

4.1 Batkube architecture

TODO

4.2 API implementation

4.3 Time hijack

TODO

4.3.1 batsky-go

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
```

— 5 —
Evaluation

— 6 —
Conclusion

Bibliography

[1] A. M. Beltre et al. “Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container Orchestration Mechanisms”. In: *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. 2019, pp. 11–20.

[2] *Kubernetes Meets High-Performance Computing*. URL: https://kubernetes.io/blog/2017/08/kubernetes-meets-high-performance/ (visited on 04/21/2020).

[3] Mikyoung Lee, Sungho Shin, and Sa-Kwang Song. “Design on distributed deep learning platform with big data”. In: (2017).

[4] 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).

[5] Seetharami R. Seelam and Yubo Li. “Orchestrating Deep Learning Workloads on Distributed Infrastructure”. In: *Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning*. DIDL ’17. Las Vegas, Nevada: Association for Computing Machinery, 2017, 9–10. ISBN: 9781450351690. DOI: 10.1145/3154842.3154845. URL: https://doi.org/10.1145/3154842.3154845.

[6] Boris Tvaroska. “Deep Learning Lifecycle Management with Kubernetes, REST, and Python”. In: Santa Clara, CA: USENIX Association, May 2019.