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

— 1 —
Introduction

TODO: Make another pass on that section after everything else is redacted (or at least the soa)

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

[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.

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 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)

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

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

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

— 2 —

State of the art

This section is organized as follow. First we put Kubernetes in context, in light of the advances made in web application development. As we will see, despite these adavances in the automation of resource management many fundamental questions remain that can only be anwsered by extensive studies on computer systems. One such question is the problem of scheduling tasks on compute resources, which we briefly present. We end this section by presenting the concepts of Batsim which is a distributed system simulator especially well suited for studies on scheduling alrogithms.

2.1 Cloud 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 resource allocation. This led to the concept of containers which takes the idea of encapsulated applications further than plain virtual machines.

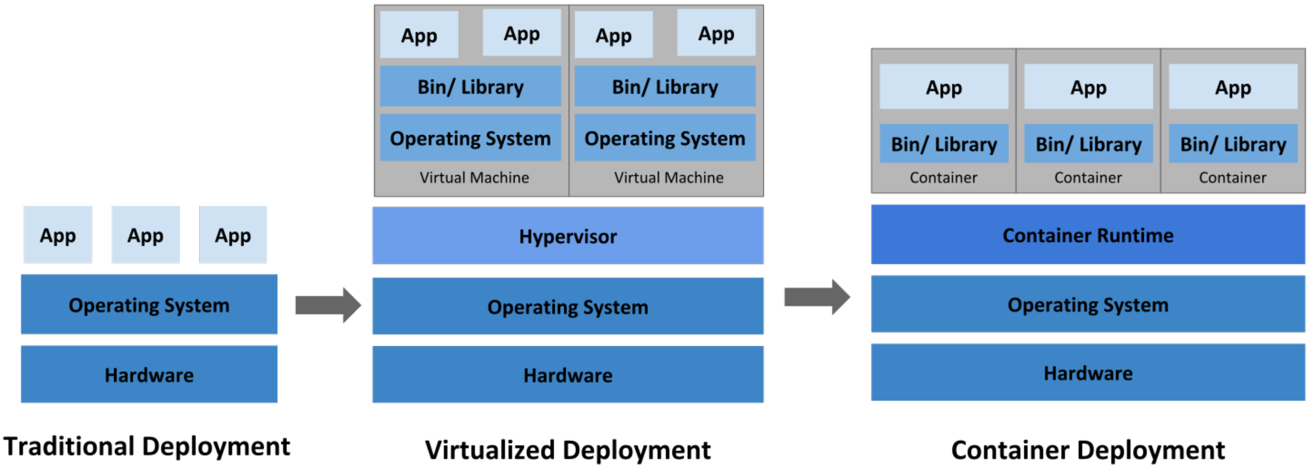


Figure 2.1 – Evolution of application deployment.
Source: *https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/*

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 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

Eventhough this paradigm enabled developing new applications with ease, many questions remain: what type of infrastructure would be best suited for my application? Would my application benefit from more cpu cores? How would different scheduling policies affect my application? Would my batch jobs compute faster with a different topology? To answer these interrogations one must conduct studies to experiment with different configurations.

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 of the infrastructure. Some systems are centralized like HPC and Data Centers, others are meant to be used from a distance like Cloud Computing infrastructures and others are decentralized like Grid Computing, 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.

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 parts are mostly built upon [6] and [3] and are aimed at depicting the current landscape of experimentation on distributed systems.

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

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

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

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

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, however it poses major scalability and reproducibility issues.

Experiments conducted on real systems may prove difficult to reproduce, as one must have access to the same system to reiterate it. Even then, changes to the infrastructure hardware and software environment diminish the chances of getting the same conditions. One solution to this problem is running *in vitro* studies, that is to say run an emulation of the system (virtual machines or a network emulation for example). This resolves the issue of reproducibility, however the matter of the cost in energy and time remains (if anything, emulation aggravates these costs).

This cost is exacerbated by the many iterations of a same experiment one must conduct in order to get statistically significant results. Workloads submitted by real users can last from hours to months and have substantial costs in energy: the means required to run them are too great and research to optimize or simply study these systems can not justify this waste of resources. For all these reasons scientists resort to simulation to study these computing infrastructures.

in silico studies, or simulation

When running simulations two primary concerns are accuracy and scalability. Accuracy is the measure of the bias between the simulated trace of an execution of an application and its trace as if it were executed on a real system (the lower it is, the higher the accuracy). Scalability is the ability of the simulator to compute simulations quickly, or run large scale experiments.

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.

list of simulators

- 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.

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

2.3 The scheduling problem

In particular, this work is targeted at experimenting with scheduling in a distributed system driven by Kubernetes. Here we present a general definition of scheduling, and the challenges it tackles.

> **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, or single jobs submitted over time by users in an unpredictable manner (as it is most often the case with HPC for example). 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 so schedulers have to implement compromises or focus on what the user requires from the system.

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
- User defined 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]).

2.3.1 How to study this problem in particular

Some simulators are especially well suited for this. Aléa and Accasim, and Batsim, which is what we build upon.

2.4 Batsim concepts

Batsim[4] is a distributed system simulator built upon the SimGrid framework. Its main objective is to enable the study of RJMS without the need to implement a custom simulator, by providing a universal text based interface. In this section we first present the SimGrid framework and then we present some of the core concepts and objectives of Batsim.

SimGrid

To understand Batsim's paradigms and view on simulation, we first need to present Simgrid's paradigms. As the latter is the framework that Batsim builds upon, Batsim and Simgrid views on simulation cannot be distinguished.

TODO

Batsim

Batsim is entirely deterministic so at to make the studies easily reproducible. Its event-based models will provide the same results given the same inputs and decision process. One other way Batsim facilitates reproducibility is through its user-defined inputs. Unlike other HPC or grid computing simulations that run on existing application traces, Batsim takes a user defined workload as an input. As a consequence, the user has no concerns such as intellectual property on application traces and may provide all his experiments materials and environment. Another advantage of this system is that the user can adapt the workload depending on its needs, to achieve different levels of realism.

Batsim, just like SimGrid, aims at being versatile. The common belief is that specialization is the key to achieving realistic results, however according to SimGrid this versatility is all but an obstacle to accuracy[3]: it is on the contrary the key to their results which are both scalable and accurate. Batsim computation platforms are SimGrid platforms meaning that theoretically, they may be as broad as SimGrid allows it. In reality any SimGrid platform is not a correct Batsim platform. Because Batsim aims at studying RJMS software, it requires a **master** node that will host the decision process. The other hosts (or computational resources) will have either the roles of **compute_node** or **storage**. Still, the user may study any topology he wishes using SimGrid models.

Thanks to its own message interface based on Unix sockets, Batsim is language agnostic which means that any RJMS can be plugged into it as long as it implements the interface.

TODO: why batsim?

Because it was developed at the lig and they want to expand its capabilities. It is language agnostic to very convenient to work with.

— 3 —
Integrating the simulator into Kubernetes

Problematic

Batsim is able to run simulations of any distributed system, to study any event-based scheduler that would implement its message protocol. Kubernetes is a piece of software where all its component, including the scheduler, revolve around a central API. Everything is then asynchronous as the API can be accessed anytime by any component.

The question that arises is, can we adapt Batsim to make it support Kubernetes schedulers? Is it possible to implement an adaptive layer between a synchronous event based simulator like Batsim and a scheduler implemented following the asynchronous paradigms of APIs?

It will follow that in order to do so, we re-implemented an API following Kubernetes specifications and intercepted the scheduler's time to synchronize it with the simulation time. This allows us to run lengthy workloads in seconds using a scheduler otherwise supposed to rely on "real" machine time. We first describe some technical concepts about Kubernetes and Batsim, and then describe how we re-implemented the API, intercepted the time, and handled the synchronization of the different times between Batsim and the scheduler.

3.1 Batsim concepts

A Batsim simulation is divided into two processes: Batsim itself and the decision process (the scheduler). Both process exchange via the ZeroMQ request-reply pattern[1]. As a consequence, the scheduler must be event based and implement Batsim's Protocol.

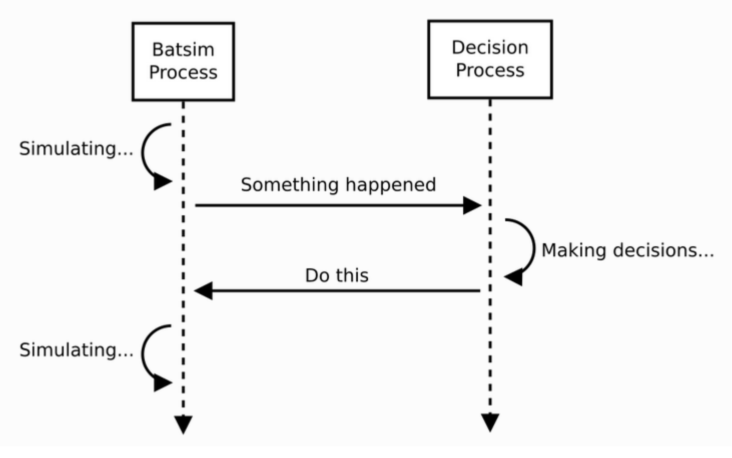


Figure 3.1 – Exchanges between Batsim and the scheduler
Source: *https://batsim.readthedocs.io/en/latest/protocol.html*

[1] `http://zguide.zeromq.org/page:all#Ask-and-Ye-Shall-Receive`

```
{
  "now": 1024.24,
  "events": [
    {
      "timestamp": 1000,
      "type": "EXECUTE_JOB",
      "data": {
        "job_id": "workload!job_1234",
        "alloc": "1 2 4-8",
      }
    },
    {
      "timestamp": 1012,
      "type": "EXECUTE_JOB",
      "data": {
        "job_id": "workload!job_1235",
        "alloc": "12-100",
      }
    }
  ]
}
```

Figure 3.2 – Example of a Batsim message

Batsim messaging interface is based on its protocol. Each message is composed of the current simulation time, as well as a list of events either from Batsim to the scheduler, or from the scheduler to Batsim. Figure 3.2 depicts a standard message sent from the scheduler to Batsim.

Batkube's features being very basic because we focused on building a working proof of concept rather than a fully fledged Kubernetes simulator, we only consider a subset of these messages that we briefly present here. More information on Batsim's protocol is available on Batsim documentation[2]

From Batsim to the scheduler

SIMULATION_BEGINS contains mostly information about the available resources in the cluster, with Batsim's configuration.

SIMULATION_ENDS is sent at the very end of the simulation: all jobs have finished, and no more jobs are left in the queues. Batsim exits this message.

JOB_SUBMITTED notifies the scheduler that a new job has been submitted. It contains information about the job type, id and specifications. We only consider jobs of type *delay* to simplify the models. Delay jobs specifications boil down to the delay length, to which we add resource requests.

JOB_COMPLETED notifies the scheduler that a job has ended, specifying the reason for it. We only consider situations where all jobs complete correctly. Their state is then always COMPLETED_SUCCESSFULLY in our case.

REQUESTED_CALL is an awnser to a CALL_ME_LATER event sent by the scheduler.

[2] https://batsim.readthedocs.io/en/latest/protocol.html

From the scheduler to Batsim

CALL_ME_LATER is an incentive from the scheduler for Batsim to wake up at a certain timestamp. When the timestamp is reached in the simulation, Batsim will send a REQUESTED_CALL to the scheduler. In our case, this particular exchange will serve as the base for time synchronisation between the scheduler and Batsim.

EXECUTE_JOB is sent when the scheduler has made a decision. It contains the id of the job at stake and the id of the resources it has been scheduled to.

Bidirectional

NOTIFY is used to send some information to the other peer. In our case, we use the NOTIFY containing no_more_static_job_to_submit to determine if the simulation has ended: knowing that there are no more jobs susceptible to be scheduled allow us to fast forward to the end of the simulation, thus saving execution time.

Batsim's output takes the form of a csv file containg information about the jobs executions. Mainly we take interest in their submission time, execution time and waiting time. Again, a detailed list of Batsim outputs can be found on the documentation[3]. During our experimentations with Batkube we interest ourselves in two metrics that can be computed from this output:

- The *makespan*, which is the total length of the simulation. It is defined as the timestamp at which the last job finishes executing, minus the origin (in this case, zero).

- The *mean_waiting_time*, which is the mean time the jobs spent waiting for a scheduling decision. The waiting time is defined by the duration between the submission time and the starting time (here the starting time is equivalent to the time at which the job was scheduled. We will see later that these two times do not correspond in Kubernetes.)

3.2 Kubernetes concepts

Kubernetes is a large ecosystem that we can not possibly describe or study in its entirety in the short amount of time that was allocated to this project. Instead, let us focus on some details about scheduling in Kubernetes and how it differs from the Batsim approach.

The basic processing unit of Kubernetes is called a **pod** which is composed of one or several containers and volumes[4]. The type of application they contain vary depending on the context: in a web platform context a pod most often hosts a service or micro-service that must be available at all times, in opposition to a batch processing context where it runs an application that is to be executed in a finite amount of time. Pods are bundled together in **nodes** (figure 3.3) which are either physical or virtual machines. They represent another barrier to pass through to access the outside world which bundles pods under the same network to facilitate communication between them, and enables the use of proxies to access the underlying services. A set of nodes is called a **cluster** which is the highest abstraction layer in Kubernetes.

Nodes take the idea of containerisation further than plain containers by encapsulating the already encapsulated services. Each node runs at least one pod, the **kubelet**, which is a process responsible for communicating with the rest of Kubernetes. More precisely, the kubelet communicates with the **kube-api-server** which is responsible for the whole cluster. This API server, as well as the other components of the **Control**

[3] https://batsim.readthedocs.io/en/latest/output-jobs.html

[4] Because of their transient nature, containers can not store data on their own. A volume is some storage space on the host machine that can be linked to containers, in order for them to read and write persistent information.

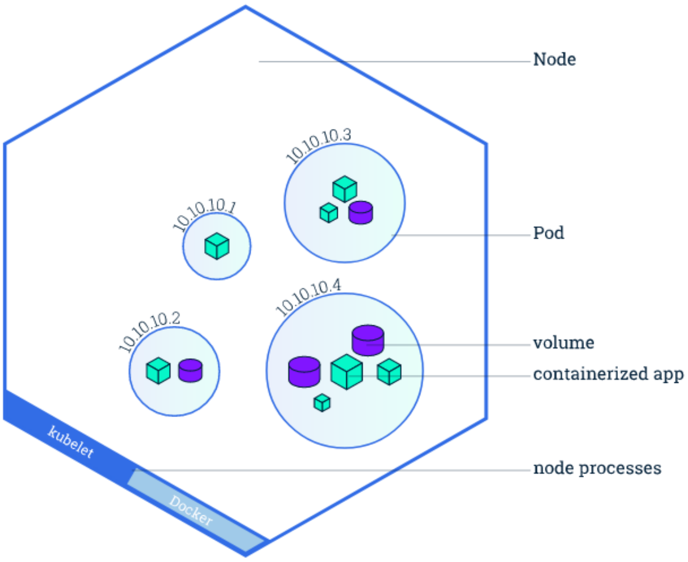


Figure 3.3 – Node overview
Source: *https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/*

Plane (figure 3.4), can be run on any machine but for simplicity they are set up on the same machine at start up. This machine is often called the **master** node and typically does not run any other container.

As stated before Kubernetes revolves around its API server which is its central component. All operations between components go through this REST API. These operations take various forms like user interactions through the commande line interface **kubectl**, scheduling operations or management of cluster data on **etcd**. We then decided to re-build the API in order to simulate any cluster to – almost – any Kubernetes scheduler.

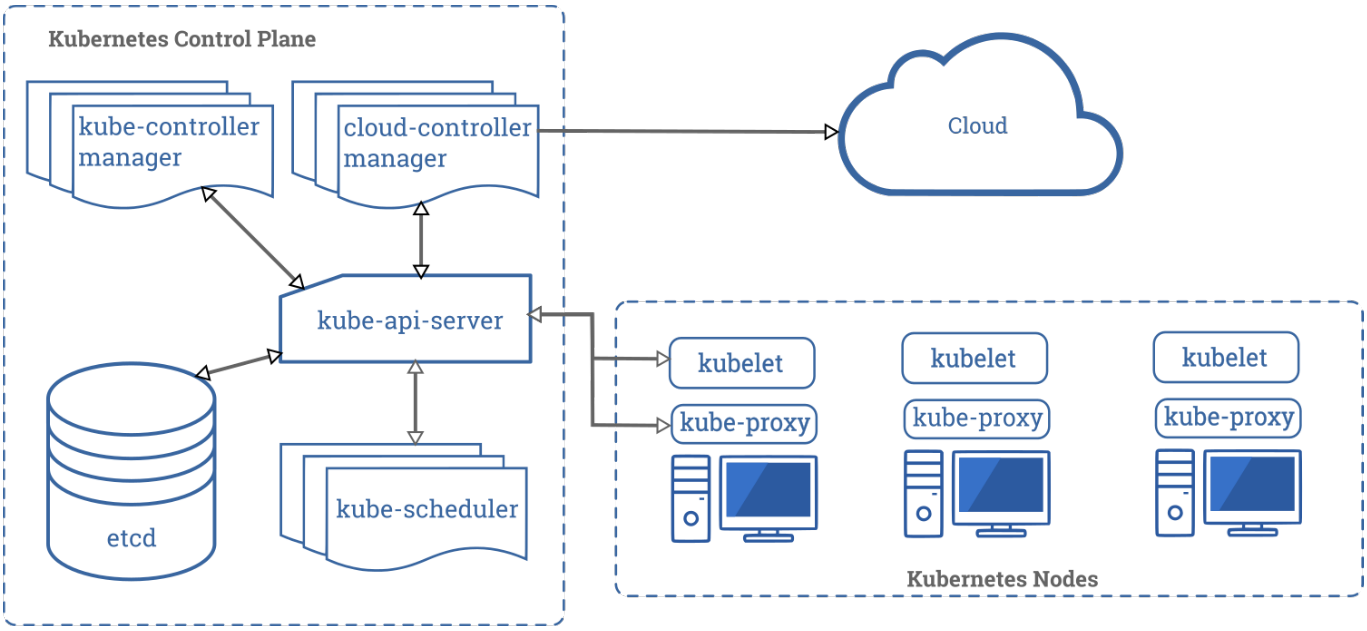


Figure 3.4 – Components of Kubernetes
Source: *https://kubernetes.io/docs/concepts/overview/components/*

3.3 Re-building the API

TODO: Explain hy this is the best approach we could find and how we did it.

3.4 Translation and resource management

TODO: maybe move this section under the last section. This is a minor problem and was really not too dificult in our case. Maybe explain why we did not translate batsim jobs to kubernetes jobs but rather directly to pods.

Explain how dirty the resource management system is (non thread safe, stored in memory, little hacks for the resource version) and briefly write on how it induces problems on the scheduling (overlap of jobs on the same resource where there is no room for them) (we talk about this in the evaluation part).

3.5 Time interception

TODO: explain how channels work briefly, to understand the algorithms.

Algorithm 1: Requester loop

```
Input: req: request channel, res: result channel map
while Batkube is not ready do
    wait
requests = []request
while req is not empty do
    m = <- req /* Non blocking receive                                        */
    requests = append(requests, m)
sendToBatkube(requests) /* Only requests with duration > 0 are actually sent.
    Batkube will always anwser.                                               */
now = responseFromBatkube()
for m in range requests do
    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
if requester loop is not running then
    go runRequesterLoop() /* There can only be one loop runing at a time     */
id = newUUID()
m = newRequestMessage(d, id) /* Requests are identified using uuids           */
resChannel = newChannel()
res[id] = resChannel /* A channel is associated with each request             */
req <- m /* The code blocks here until request is handled                     */
now = <-resChannel /* The code blocks here until response is sent by the
    requester loop                                                            */
return now
```

3.6 Time synchronization

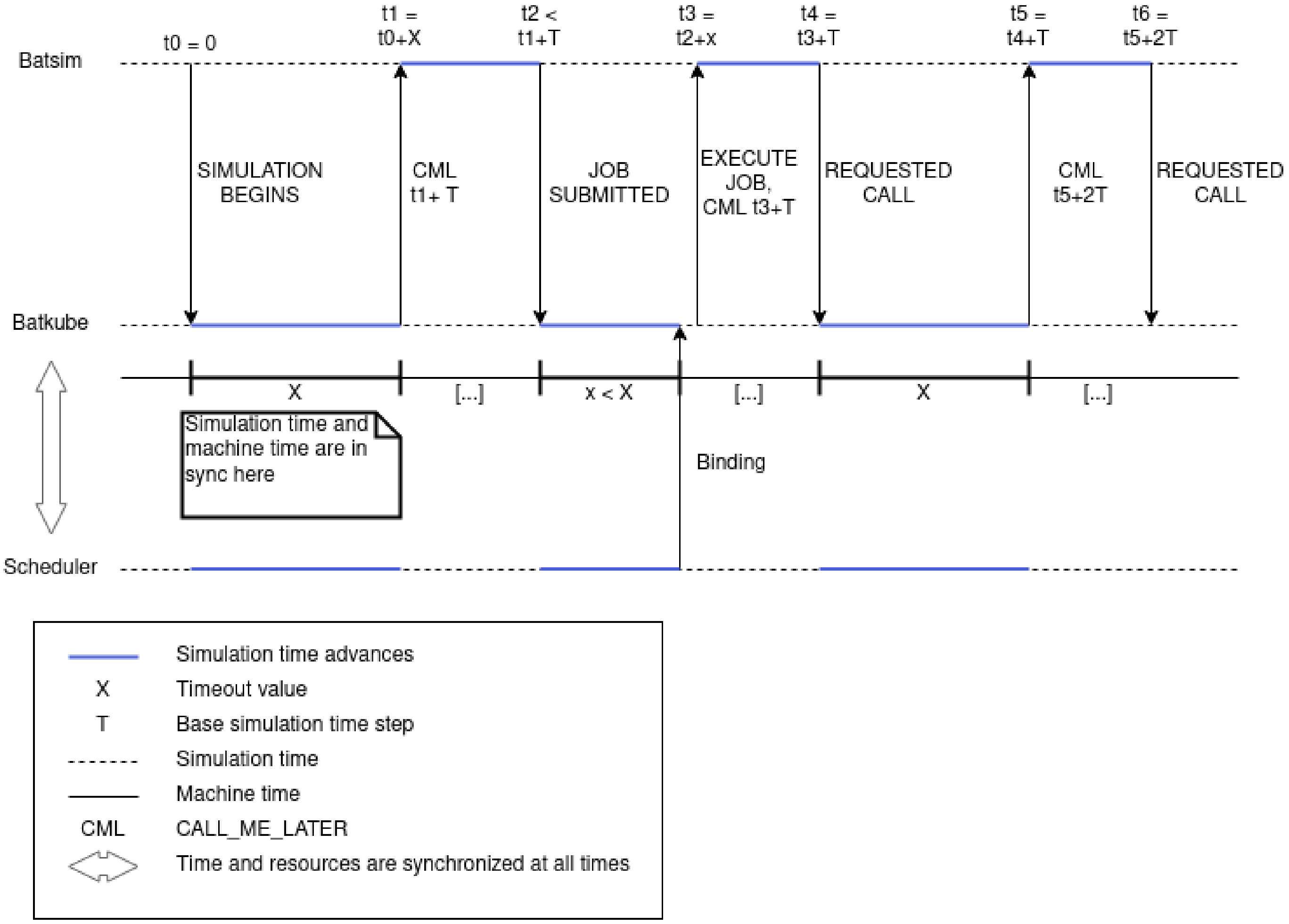


Figure 3.5 – Time sync between the three components. The broker has to take into account both machine time and simulation time.

— 4 —
Evaluation and discussion

Because SimGrid has already been thoroughly tested and validated, we do not need to run extensive experiments to validate Batkube simulation models. Moreover, since we only consider simple delay jobs, validation is not really necessary. Still, even though the underlying models are sound, Batkube adds a considerable overhead to Batsim because of the time synchronization between the simulator and the scheduler. We want to verify to what extent time manipulation impacts the scheduler behavior, and also that Batkube's fake Kubernetes API mimics the real API well enough to let the scheduler run as expected.

In the next sections, we present the workloads and platforms we chose to study, how we conducted experiments on a real cluster, and a study on Batkube's parameters and their effect on the outputs.

4.1 Experiments environment

The entirety of the experiments are done with the default Kubernetes scheduler **kube-scheduler** release **v1.19.0.rc-4** (commit 382107e6c84). This choice was made because it was the scheduler used during development, and because supporting another scheduler would mean more development time which we could not afford. Still, it is sufficient to experiment with the simulator and verify the scheduler's behavior in the simulation.

4.1.1 Real experimental testbed

In order to validate the simulator results we then need to compare it against workloads run on a real cluster. For reproducibility and simplicity sake, we choose to validate the simulator with an emulated cluster run in containers.

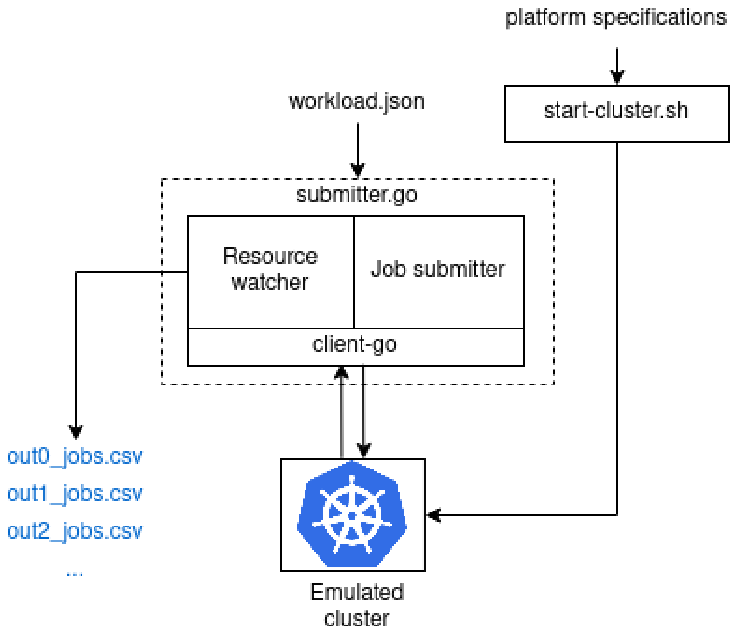


Figure 4.1 – An emulated experiment.

Figure 4.1 illustrates how this is done. First, we create a k3s cluster run using docker-compose – `start-cluster.sh` is a helper script made for this. Then, a Go script which takes a workload as an input submits the jobs at the right time and writes the ouputs to csv files – which have the same format as Batsim's csv output files. We run each workloads 10 times in order to get statistically meaningful results, except for the realistic workload which we only run one time because it is already 10 hours long.

The emulated cluster is limited in terms of variety and capacity. First, `start-cluster.sh` only allows the nodes to have the same amount of available cpu, memory or storage because there was no need for any complex system for our experiences. Secondly, the maximum amount of cpu, memory or storage we can make available for each node is capped to the host system capacities. For example, if the host system possesses 8 cpu cores, the nodes will have a maximum of 8 cpu available. This will have implications when trying to run workloads recorded on real systems: either we get to find a workload that complies with the host system capacity (which is very unlikely), or we adapt the workload so the jobs requirements do not exceed the host capabilities (see section 4.1.2).

4.1.2 Studied workloads

We consider three workloads, representing three different situations. The first two are simplistic and very controlled, and the last one depicts a more realistic case. In all cases the required resources are only quantified in cpu only to simplify the study. Note that Batkube does support memory requets, we just do not wish to add this other layer of complexity to our experiments.

- A *burst* workload, consisting in an important amount of jobs submitted at once. 200 delays with duration 170s and requesting 1 cpu are submitted at the origin.

- A *spaced* workload, where jobs of the same nature are submitted at regular intervals. 200 delays with duration 170s, and requesting 1 cpu are submitted every 10s.

- A *realistic* workload, which is extracted from a larger trace of a real system.

The first two workloads are straight forward and could be generated with the use of a plain text editor (understand `vim` and its macros). The third workload required more processing to be obtained.

Standard Workload Format processing

First, a trace in standard workload format (swf) was obtained on a web archive[1]. The chosen workload was `KIT-FH2-2016-1.swf` because it is the most recent and is relatively lightweight. Secondly, `evalys`[2] allowed us to extract a subset of this workload lasting for a given period of time and with a given mean utilization of the resources. We chose a period of 10h with 80% utilization of the resources so as to keep reasonable experiment durations – Later on we experiment with larger workloads to test out Batkube's limits in terms of scalability. The third step is translating this extracted workload to a `json` file that can be read by Batsim, which is done with a script written in Go.

After extracting this subset, we are left off with a workload containing jobs spanning up to 45h and using up to 24048 cpu (or cpu cores), which is undoable at our scale on our emulated cluster. We need to trim job durations as well as cpu usage, as we are limited in cpu by the host machine. This is done during the translation to the `json` format. The durations are trimmed down to a maximum of one hour and the cpu usages are normalized so the maximum amount of cpu requested equals the amount of cpu available per node on the host machine. Otherwise, the job would be unschedulable which would not present much interest.

4.1.3 Studied platforms

The platform used for the first two workloads, *burst* and *spaced* is composed of 16 nodes each heaving one cpu. For the *realistic* workload however, we use a single node composed of six cores for the following reasons.

First, the host machines where the experiments were conducted had six cpu cores available. This means that if we want to be able to run an emulated cluster equivalent to this platform we can't exceed six cpu per node. We use the maximum amount of available cores in order so as not to obtain too low values when normalizing the resource requests on the jobs. Indeed, Kubernetes only allows for a precision of 1 milicpu, so any value bellow that is not considered a significant number. Normalizing on six cpus instead of one allows us to get more significant number. Also, only one node gives us a satisfactory overall resource usage: with more than one node one resource is almost always available making the scheduling operations trivial.

4.2 Study of the simulator parameters

The simulator has a few parameters that impact the simulation speed and accuracy. The objective is to study the effects of these parameters on the simulation to better understand the scheduler behavior when running in coordination with Batkube.

The objective here is to fine tune the parameters in order to find a compromise between accuracy and scalability. We want to know which combination lead us to the most stable results, while keeping simulation time as low as possible.

The parameters are:

- The *minimum delay* we have to spend waiting for the scheduler.
- The *timeout* value when waiting for scheduler decisions.
- The *maximum simulation time step*, which is the maximum amount of time Batsim is allowed to jump forward in time.

[1] https://www.cs.huji.ac.il/labs/parallel/workload/logs.html

[2] https://github.com/oar-team/evalys

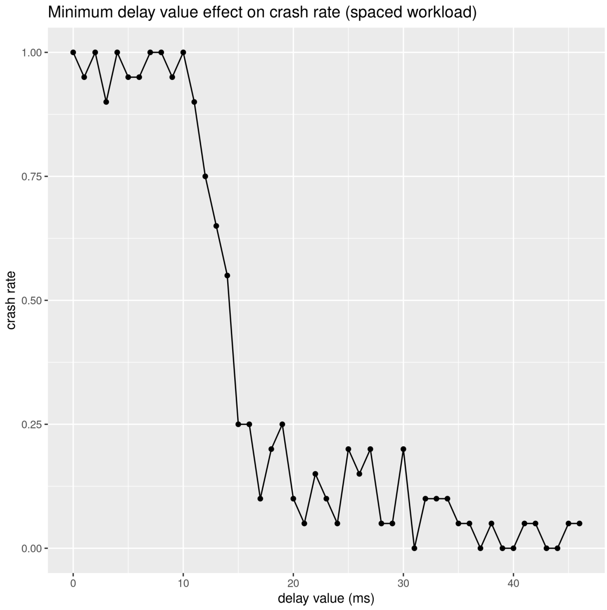


Figure 4.2 – Crash rate of the simulations against minimum delay.

We first study these parameters one by one by fixing the other parameters to some other value, then we study what effects these parameters have in respect to one another, and finally we conduct scalability experiments to test Batkube's stability and performances on large workloads.

4.2.1 Minimum delay

Earlier in the development of batkube we noticed that not leaving enough time to the scheduler each cycle lead it to crashes and deadlocks, ultimately failing the simulation. This time is independent from any decision making we would receive from it which is why it is called *minimum wait delay* instead of a plain *timeout* - which is in fact another parameter we will study later.

For each workload, we compute the crash rate every 5ms, from 0ms to 50ms. Each point is made by running the simulation 15 times and recording the exit code as well as the simulation time. The other parameters are: `timeout=20ms`; `max-simulation-timestep=20s`. As we will see later those do not offer acceptable simulation results but they allow us to run prompt simulations, as accuracy do not concern us here.

As we can see on figure 4.2, the crash rate decreases dramatically as soon as the minimum delay reaches a certain threshold, which here is 10ms. This crashing issue though was resolved with an update of the scheduler : the success rate flattens out at 100% – or around 100%. Then, with earlier versions of the scheduler, the user may have to adjust the minimum delay in order to run simulation smoothly.

We also observe on figure 4.3 – which was made with an updated scheduler – a prompt increase in simulation time from delay value 20ms. This is due to the fact that the *timeout* value is 20ms, which is reached most of the time because the vast majority of the calls to the scheduler do not result in a decision making. After this value, we notice a direct correlation between *minimum delay* increase and simulation time increase. It follows that the best choice for the *minimum delay* now is zero, and we will use this value for the rest of the experiments.

4.2.2 Timeout

This value is the maximum amount of time we leave for the scheduler to react. A `timeout value` not large enough may lead to inaccuracies in the simulation: for example, if the scheduler needs 30ms to make

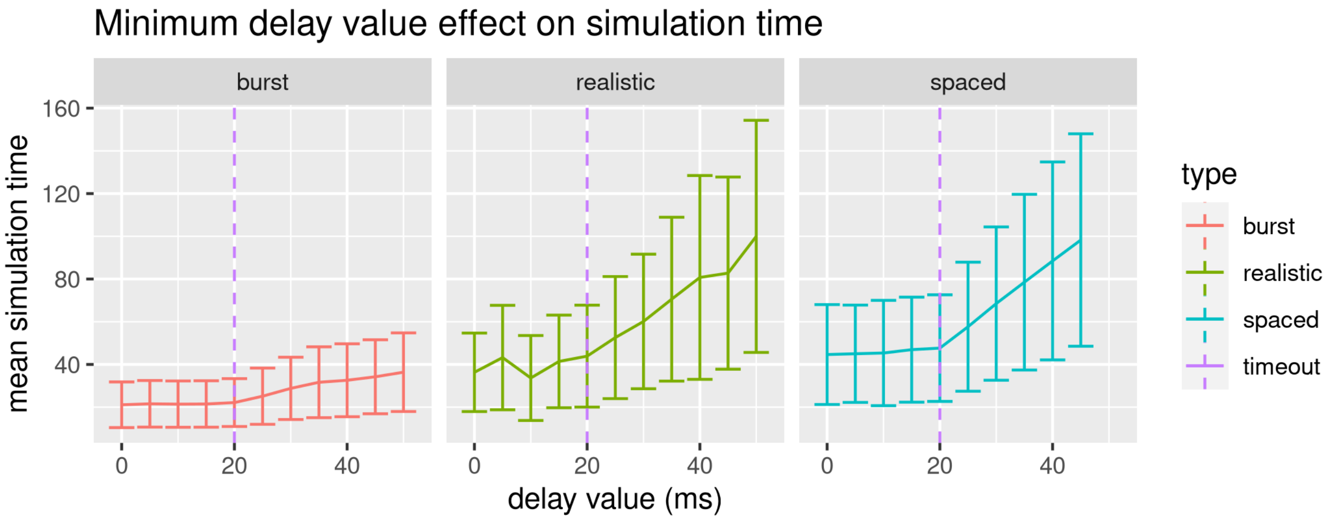


Figure 4.3 – Mean duration of the simulations (in case of success) against minimum delay. Error bars show confidence intervals at 95%.

a decision upon reception of a message, and the value of the timeout is 20ms, Batkube will receive the decision on the next cycle which may happen several dozens of seconds later (depending on the *maximum simulation time step* value). On the other hand, a *timeout value* too large will induce longer simulation times unnecessarily. Indeed, once the simulator was given enough time to process a message, any time following is spent idling. We want to measure which `timeout value` is just enough for the scheduler to be able to make a response without spending any time idling.

We run each workload with a `timeout value` ranging from 0ms to 100ms, with a step of 1ms. Each time we measure the duration of the simulation as well as the makespan and the mean waiting time. The latter two will enable us to compare the results against the emulated results in order to estimate the accuracy of the simulation. The other parameters are set to: `min-delay=0ms`, `max-simulation-timestep=20s`

TODO: redo the simulations for the realistic wl, removing some values of timeout and adding repetition to show aggregated metrics. (non aggregated ones are too dispersed)

TODO: 4.4a does not have the same scale on the x axis as the others.

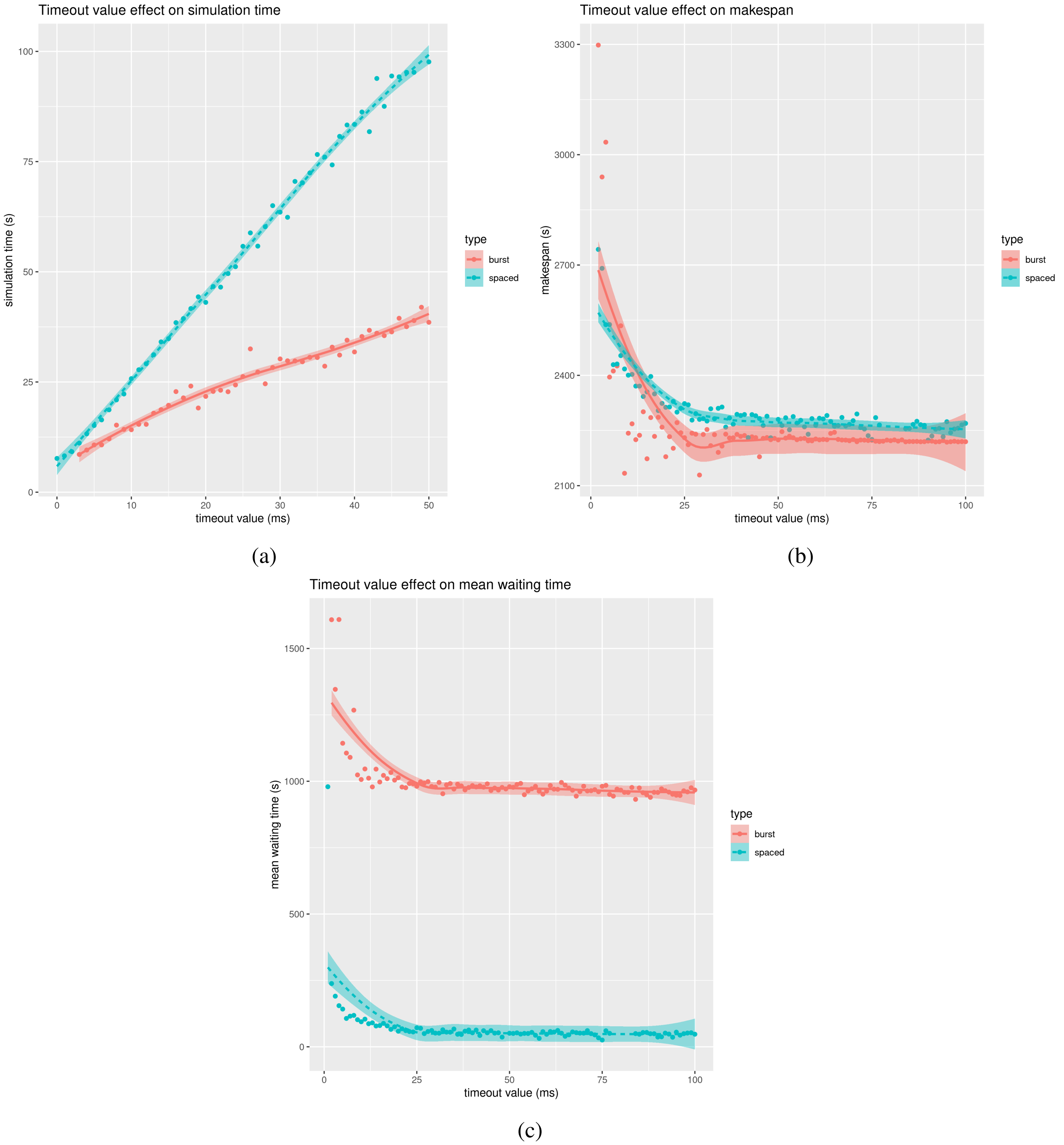
TODO: Gantt charts to show the gaps.

As we expected, a *timeout value* too low results in the scheduler missing a few cycles each time it wants to communicate a decision making, thus increasing the makespan and mean waiting time. As the *timeout* increases, it reaches a point where the scheduler consistently sends decisions in the same cycle as the one where it has received the message that triggered the decision making. After this point though the curves keep decreasing, showing that the gaps keep receding afterwards. However, the gain in accuracy is shallow and considering that there is, again, a direct correlation between the *timeout value* and the simulation time, it is desirable to keep this value at the limit where the results start to stabilize. In this case, according to figure 4.4b, `timeout-value=50ms` seems like a decent compromise between accuracy and scalability.

With such simple workloads and platforms, the decision making time is very low (it is but a matter of milliseconds), but we can imagine it may reach much higher values given a bigger platform and more complicated workload.

4.2.3 Maximum simulation time step

Having a high maximum time step value will allow Batsim to jump forward further in time. This may result in skipping scheduler decisions that could have been made in the mean time, delaying them to when Batsim decides to wake up. We expect increasing this value to have an analogous effect to the timeout value: higher



(a)

(b)

(c)

Figure 4.4 – Effect of the timeout value on the simulation

simulation speed, but also decreased accuracy due to gaps (delays) in the decision process.

PROTOCOL: same as timeout. max timestep ranges from 1s to 100s with a logarithmic scale (1s, 2s, . . . , 9s, 10s, 20s, . . . , 90s, 100s) Other parameters: timeout=20ms; min-delay=0ms.

TODO: basically the same graphs as in the timeout

4.2.4 Parameters inter dependency

Studying the parameters independently is not enough, we need to study their impact relatively to each others.

For instance, the *maximum simulation time step* and the *timeout* value are tightly linked together regarding their effect on accuracy. Both decreasing *timeout* and increasing *maximum simulation time step* will increase the amount of delays in the decision making of the scheduler, but also their length, multiplying the impact on accuracy.

On the other hand, we do not except the *minimum wait delay* to have any impact other than increasing the simulation stability.

PROTOCOL: We know the simulations are statistically the same by now, and we're confident enough to run only one simulation per point. Timeout value ranges from 5ms to 50ms, max timestep ranges from 1s to 100s (logarithmic again).

TODO: Facet graphs: timestep vs timeout vs accuracy vs simulation time. We can define accuracy as one on the euclidean distance between emulated and simulated makespan and mean waiting time:

$$\frac{1}{\sqrt{(makespan_{sim} - makespan_{emu})^2 + (waitingtime_{sim} - waitingtime_{emu})^2}}$$

4.3 Validation of the simulator outputs

TODO: With default parameters (timeout TBD with experiments results, max time step same, min delay 0), compare simulated and emulated results.

Here : the Gantt charts from evalys.

Study on two metrics : makespan and mean_waiting_time. Show the box plots for simulated and emulated metrics.

Discussion:

Container pull and startup time not accounted for in the simulation.

The scheduler over allocates when it should not, reducing makespan.

The scheduler sometimes does not seem to get update on nodes resource state and realizes late that some nodes are free (is this still the case? need to plot gantt charts)

4.4 Scalability experiments and scheduler limits

We have extracted a few workloads recorded on real systems to test out the scheduler performances in regard to scalability, that is to say the ability to run large workloads in a minimal amount of time.

PROTOCOL: go all out on large workloads and platform

TODO: can't really run simulations that are too long because the scheduler fills the memory pretty fast.

.1 Reproducing the experiments

TODO: organize this part (for now everything is simply copy pasted here) and complete it

The command used to run the scheduler is

```
./scheduler --kubeconfig=<kubeconfig.yaml>
--kube-api-content-type=application/json --leader-elect=false
--scheduler-name=default
```

Only the path to the kubeconfig.yaml changes to either point the the emulated or simulated cluster.

Batkube is run with

```
./batkube --scheme=http --port=8001
```

followed by the simulator options.

Batsim is run with option enable-compute-sharing: for a reason unknown, Kubernetes scheduler tends to over allocate resources in some cases (especially with smaller jobs) which makes Batsim crash if this option is disabled. We must allow compute sharing even when it is not expected in order to capture the scheduler behavior as precisely as possible.

Those are the Batkube options that did not vary during the experiments:

- backoff-multiplier: 2 (default value)
- detect-scheduler-deadlock: true. Obligatory for automating experiments
- fast-forward-on-no-pending-jobs: the scheduler is not susceptible to reschedule running jobs (there is a de-scheduler for that) so we might as well fast forward when there is nothing to schedule.

The option scheduler-crash-timeout did vary between experiments to make up for odd scheduler crash detections (it was increased up to 30s). However, it did not have any impact on the results as we do not take into account simulation time in case of failure.

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]`.