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.

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[21] 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[22]. Very often simulators are implemented at the same time as new schedulers or *Resource and Jobs Management Systems (RJMS)*[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 [22], 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[11] which is a general

---

[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.

purpose infrastructure simulator with modularity and separation of concerns in mind. Batsim is based on SimGrid[9] 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 —

Background and related work

2.1 Studying computer infrastructures

Even though the containers 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 (which 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[1] 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[14] that can hardly be predicted.

Scientist still have tools to study those systems. More precisely, there are 3 options as described in[19]: *in vivo*, *in vitro* and *in silico* studies, which correspond respectively to experiments on real testbeds, emulation and simulation.

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 in the first place. Event in the eventuality where access to the exact same system would be possible, changes in the infrastructure and software environment over time would diminish even more the chances of recreating a sane environment. Moreover, studying new algorithms imply testing them under a

---

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

wide range of systems (under different system size, topology, and hardware) which is impossible to do with a real computer as we can not change the hardware of those infrastructures.

One solution to these issues is running *in vitro* studies, that is to say run an emulation of the system. This resolves the issue of reproducibility, however the matter of the cost in energy and time remains. If anything, emulation aggravates these costs: emulation adds a consequent overhead to the system and running the same workloads in reasonable time requires access to a system of (at least) equivalent size. 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.

in silico studies, or simulation

The answer to these scalability considerations is simulated infrastructures. Simulation allows scientists to conduct experiments or thought experiments that would otherwise not be possible in the real world. One can think of simulations of the universe, prediction models for the weather or modeling some microbiome in biology. Computing itself makes no exception and researcher have created models of computer systems in order to experiment with new scheduling policies, network topologies or planning for systems capacity, for example. Simulation dramatically reduces experimentation cost and allows for reproducibility. Workloads on supercomputers may span weeks or months, whereas a single standard laptop can simulate this same workload in a matter of seconds or hours, depending on the simulator. More importantly, other scientists would need access to the same system the experiment was run on to reproduce the experiment whereas a simulator supposedly brings the same output regardless of the device it is run on. The only caveat that remains in terms of reproducibility lies in the application traces used to run off-line simulations that contain critical data concerning the application it was generated with.

However, even though simulators theoretically allow for effortlessly reproducible experiments, the way they are developed sometimes make them hardly usable. Indeed, lots of simulators are only intended to be used only by their developers, in order to validate the results of one particular work or paper. These simulators are generally unreleased, and when they are, they end up unmaintained. As a consequence, the experiences they served on are *in fine* just as difficult to reproduce as the experiences conducted on real systems. Another issue that this induces is that simulators are specialized to tackle one specific problem, and end up being built upon over-fitted models to validate this problem. These simulators cannot make sense in any other context and therefore can not be re-used for any other project.

There is a common belief that specialization is necessary to obtain the most accurate results as possible on a given problem. SimGrid [9] however is a framework for simulation that builds on the idea that versatility serves accuracy, while ensuring high scalability.

TODO: add some ref to domain specific schedulers that would not be covered by the next sections.

2.2   SimGrid

SimGrid [9] is a framework for builing distributed systems simulators and is written in C. It uses simple analytical models for all its resources to ensure high scalability, and also because those models proved accurate. For instance, a task execution time boils down to a compute cost divided by the compute speed of the resource it is allocated to. Just like CPU which are not modeled at the cpu cycle level, network transfers and disks are not modeled at packet level and block level either. It was developed – and improved – with versatility in mind, which according to its makers was the key to its excellent performance both in accuracy and scalability. This *versatility* is not to be confused with *genericity*: their models provide some amount of genericity so that they can be improved for specific tasks, but they are versatile enough to cover various of domains, each accurately.

Projects based on SimGrid span accross a wide range of domains within distributed systems. WRENCH [8] is a *Workflow Management System* library that provide a high level API to eable the study of WMS. Simbatch [7] is a (discontinued) batch systems simulator made for the study of batch schedulers. Work has been done on the side of Volunteer Computing as well with SimBOINC [16] that has been discontinued as well due to major reconstructions of SimGrid and the BOINC client or SimGrid-BOINC [10] by some of the same people behind SimGrid. And of course the recent Batsim [11] this work is based upon, which is a RJMS simulator, relies on SimGrid. All these projects prove the ability of SimGrid to perform in very different contexts, and SimGrid models are known for their validity and scalability.

GridSim [5] is another library for development of grid computing simulators. Contrary to SimGrid it is coded in Java and it is therefore cross platform. GridSim has a broader view on what a grid is: when SimGrid models wired networks and therefore only allows for specifications of localized resources, GridSim allows resources to be specified in any time zone. GridSim also support resources in *time-shared* or *space-shared* mode while SimGrid only models time shared resources. Modeling space shared resources allows GridSim to simulate multiple users competing to submit jobs simultaneously on the same resources, which is a feature that would require extending SimGrid models.

Eventhough GridSim is very popular in its domain, its models are not as valid as SimGrid models. According to SimGrid's team a simple inspection of GridSim's code and its follow up cloud infrastructure simulator CloudSim [6] is enough to find invalidating cases to its networking models [26]. Moreover, still according to SimGrid team, scalability tests showed that GridSim complexity is quadratic in the number of tasks and linear in the number of workers, as well as having a considerable memory footprint compared to SimGrid [9]. In comparison, SimGrid simulation time and memory usage are more stable and polynomial at best both in the amount of tasks and workers.

2.3 The scheduling problem

One notorious problem on the field of distributed systems is the allocation of queued jobs to available resources.

> **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 ([25], [21], [4]). In order to better study the effect of different scheduling policies on a system a reasearcher team at the LIG have created Batsim which is a versatile distributed system simulator built on SimGrid and focused on the study of schedulers.

2.4   Batsim

Batsim[11] 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.

It 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. Batsim also facilitates reproducibility 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[9]: 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 Batsim is language agnostic which means that any RJMS can be plugged into it as long as it implements the interface. This property allows us to plug any scheduler we wish to Batsim, including Kubernetes schedulers, which allowed us to run simulations of Kubernetes clusters.

2.5   Kubernetes in the context of 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.

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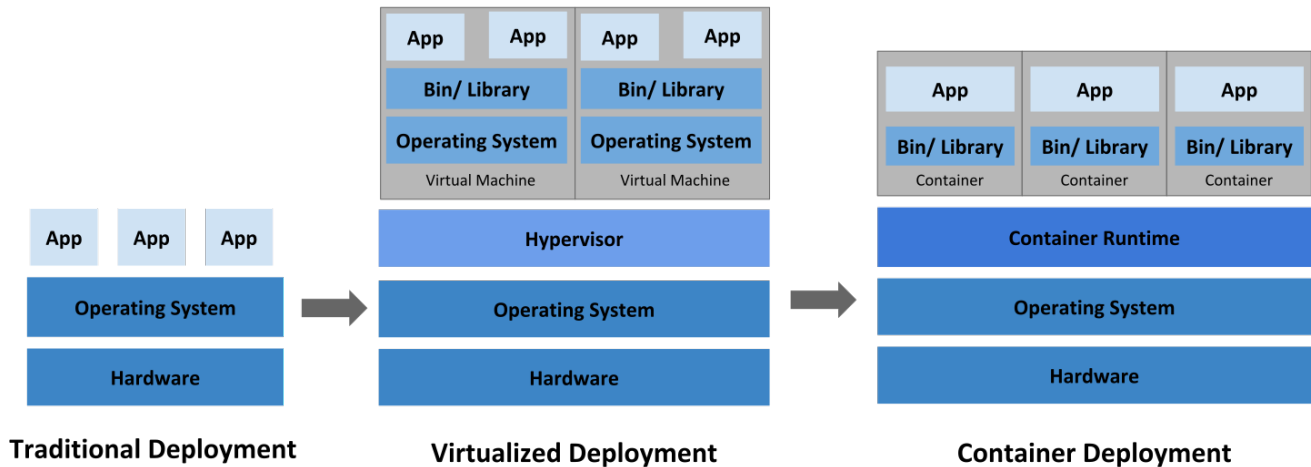
Figure 2.1 – Evolution of application deployment.
Source: *https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/*

(containers greatly facilitate integration tests).

The *Cloud Native Computing Foundation (CNCF)* [2] 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."[3]

Kubernetes[4] is the implementation of this general idea and was announced 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.

2.5.1 HPC and Kubernetes

In this work we use Kubernetes schedulers as batch schedulers because batch scheduling is our domain of origin (Batsim was developed in close relation with the teams behind the grid computing simulator SimGrid[9] and the resource manager OAR [20]), and because it is convenient for us to test our approach with HPC workloads. However, even though Kubernetes does have means to handle such workloads like batch schedulers [17] and a special type of resource[5], these two fields differ fundamentally because of the nature of the issues they tackle. We do not intend to evaluate scheduler performance but rather prove that such studies are possible with Batsim, which is why this work is conducted this way. Still, there are perspectives for Kubernetes in the HPC community and we believe HPC systems could benefit from this work.

The difference between HPC and cloud computing lies in the workloads they are intended to tackle. Kubernetes was designed to run applications. Services or micro services are run in containers and are expected to be available at all times : they are replicated as many times as desired and restarted whenever

---

[2]https://www.cncf.io/
[3]https://github.com/cncf/toc/blob/master/DEFINITION.md
[4]https://kubernetes.io/
[5]https://kubernetes.io/docs/concepts/workloads/controllers/job/

a failure occurs. High availability is at the core of Kubernetes container management. On the other hand, depending on implemented scheduling policies, HPC is focused on metrics such as user wait time, resource usage or energy cost. One example of a fundamental difference between cluster computing and batch scheduling is the handling of a task failure. In HPC it is sometimes not sufficient to restart the single job that failed and the entire submission must be re-run if it is part of several jobs computed in parallel. In cluster computing the task is simply restarted without any other consideration – other than ensuring its dependencies are restarted with it.

Kubernetes is now the standard for AI and Machine Learning as shown by the many efforts at making this coupling an efficient environment[18][24][23], 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] now support containers as well.

Indeed, containers have many advantages from which HPC users could benefit from.

- First off, research has shown that Kuberenetes offer similar performance to more standard bare metal HPC[3].

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

- 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.

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, but that time may come soon as containers are becoming more and more integrated in modern infrastructures.

2.6 Related work

Not many projects exist on Kubernetes simulation or have been disclosed. We were able to find two projects that propose simulations of Kubernetes clusters.

JoySim [28] is a fully fledged Kubernetes simulator developoed in an industrial context. Simulations are based on synthetic events and their mock nodes simply simulate a resource usage without any underlying model. The strength of this simulator is its scalability which it obtains thanks to its very light weight simulated nodes. JoySim is aimed at studying the quality of the scheduling and its performance in complex scenarios by generating metrics such as resource usage and scheduling time. Although, to the best of our knowledge, it is not suited for batch scheduling and tackles more classic issues for Kubernetes such as services availability and resource utilization. The user may specify events to occur during the simulation in order to test out different scenarios. Unfortunately, while an open source released in planned, this piece of software is currently closed source.

k8s-cluster-simulator[27] is an open source cluster simulator for evaluating schedulers, which originated from a end of studies project like this work. Like JoySim, it relies on simplistic models: pods are submitted with certain resource requirements, for a certain amount of time. One can provide any scheduler implementation they want as long as they do so through their Go interface. This reduces the amount of schedulers that can be tested to Go implementations only, unless the user is willing to implement an adaptive layer to

---

[6]`https://github.com/kubernetes-sigs/kube-batch`
[7]`https://slurm.schedmd.com/containers.html`

support other schedulers.

Two simulators are very close to Batsim in terms of capabilities and objectives, that is to say the study on scheduling policies.

Alea2[15] is a grid computer system simulator based on GridSim. Like Batsim, its *ready-to-use* philosophy make it a fantastic simulator when one desires to experiment with different scheduling policies with little set up overhead. Because it is based on GridSim, it is all written in Java and is thus cross platform. Alea2 strength lies in its modularity: its object oriented paradigms make it very easily extensible and customizable. Unlike Batsim it does not offer a decoupling of the simulator and the decision process and therefore the user will have to rely on the implemented scheduling algorithms, although these algorithms cover the most standardly used scheduling policies. The user will have to implement other policies he would like to test out. The simulator handles inputs in the form of *Standard Workloads Format (SWF)* or *Grid Workloads Format (GWF)*. Batsim proposes a script to process SWF files into its own input format but does not handle them directly.

Accasim[12] is a simulator for studying *Workload Management Systems (WMS)* in HPC infrastructure. It is similar to Alea in the sense that the decision process is not decoupled from the simulator and standard scheduling algorithms are implemented in Accasim, so that the user does not need to set up extra software in order to start experimenting. In addition, the user may provide extra data about the system status (power consumption, temperature or resource failures) in order to support advanced scheduling algorithms. These take the form of additional events transmitted to the simulator. Accasim loads jobs incrementally and cleans them upon completion which reduces its memory usage compared to Batsim, which loads everything in memory at the start of the simulation.

Both simulator outperform Batsim ([12]), especially with large workloads because of its high memory usage that hinders the simulation speed. However, both lack the decoupling between the resources and the scheduler that we need in order to plug Kubernetes schedulers onto it.

— 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.

Batkube features

TODO: Clearly state what Batkube is capable of, and what it does not do. This helps describing only the parts of the tools we use while leaving the rest for the reader to look for on official documentation.

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.
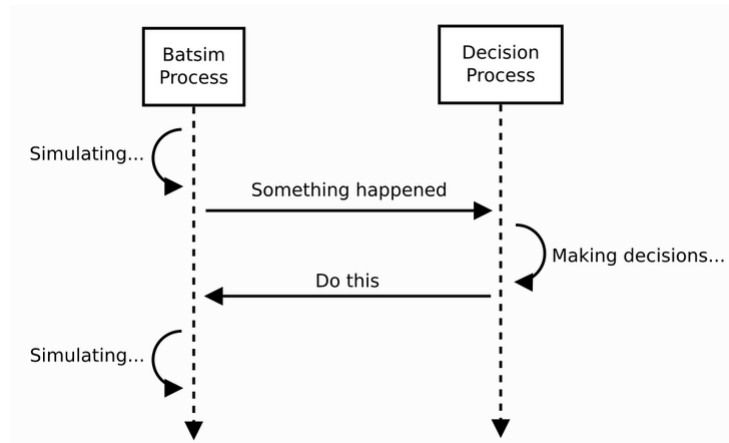
---

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

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

A Batsim platform is a SimGrid platform, defined in the xml format. A node can endorse the role of *master*, *compute_resource* or *storage*. Here we only consider master nodes which host the decision process, and compute resources to which we add our custom resource capacities. These additional fields are *core*[2] which is the amount of cpu the node has and *memory* which is memory capacity of the node. Of course, storage resources can be taken into account in future development of Batkube. Also, we do not consider the links between the nodes for now because we do not support parallel jobs. Finally, Batsim proposes an energy model that we decided to ignore as well.

Batsim takes one or several workload as inputs, which are json files containing jobs definitions. The jobs are defined with the following inputs. First, they are identified by an *id* which is unique within each workload and are submitted at a time defined by the *subtime* field. The *res* field states the number of resources each job requires, although we don't use this field because we can't specify *which* resource we require. Instead we pass resource requests as optional parameters. We support the additional fields *cpu* which has a minimum value of 100m cpu just like Kubernetes cpu requests[3] and *memory* which also complies with Kubernetes compute resource definitions. Jobs follow a certain *profile* that defines the nature of the job. These profile may be as simplistic as *delay* profile which makes the resource wait for a given amount of time, or describe parallel tasks using matrices to describe the amount of exchanges between the allocated nodes. For now we only consider delays to simplify the implementation of the simulator. Also, because Kubernetes allows the use of multiple schedulers[4], we support the field *scheduler* which contains the name of the scheduler the job should be scheduled with.

Batsim messaging interface is based on its protocol. Each message is composed of the field *timestamp* which contains the current simulation time, as well the field *events* which is a list of events either from Batsim to the scheduler, or from the scheduler to Batsim. Figure 3.3 depicts a standard message sent from the scheduler to Batsim.

Batkube's features are limited because we focused on building a working proof of concept rather than a fully fledged Kubernetes simulator. This is why only consider a subset of these messages that we present here. More information on Batsim's protocol is available on Batsim documentation[5]

---

[2]The core field is already present in the resource definition, but it is not forwarded to Batsim for unknown reasons at the time this report is written.

[3]https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

[4]https://kubernetes.io/docs/tasks/extend-kubernetes/configure-multiple-schedulers/

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

```
{
    "nb_res": 1,
    "jobs": [
        {"id":"1", "subtime":0, "res": 1, "profile": "delay10"},
        {"id":"2", "subtime":3.4, "res": 1, "profile": "delay10"}
    ],
    "profiles": {
        "delay10": {
            "type": "delay",
            "delay": 10,
            "scheduler": "default",
            "cpu": "1.5",
            "memory": "500Mi"
        }
    }
}
```

Figure 3.2 – Example of a Batsim workload

```
{
  "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 3.3 – Example of a Batsim message

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 on 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.

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 containing 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[6]. 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.)

---

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

3.2   Kubernetes concepts

Kubernetes is now a large ecosystem to which more than 2 700 developers have contributed over the years. In this section we present the fundamental concepts and the components that make up Kubernetes.
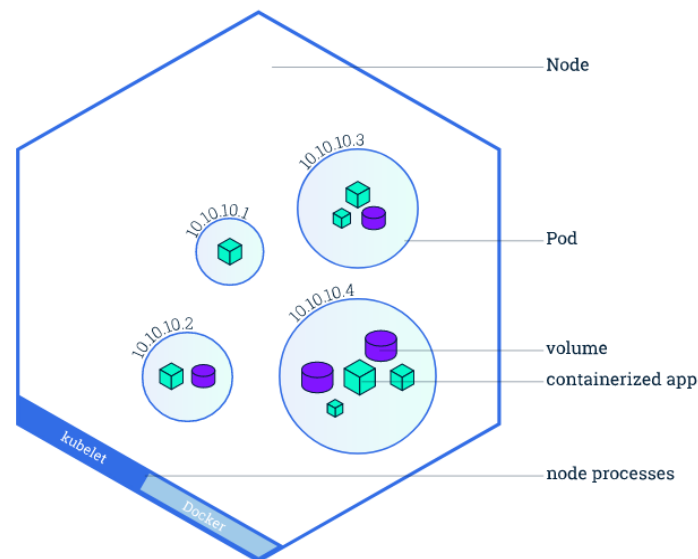

Figure 3.4 – Node overview
Source: *https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/*

The basic processing unit of Kubernetes is called a **pod** which is composed of one or several containers and volumes[7]. 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 3.4) 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 containerization 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. We refer to this API as the api-server, as it is called within the Kubernetes community. This API server, as well as the other components of the **Control Plane** (figure 3.5), 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.

TODO: Talk about Kubernetes schedulers. What they are based on, how they communicate to Kubernetes.

---

[7]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.
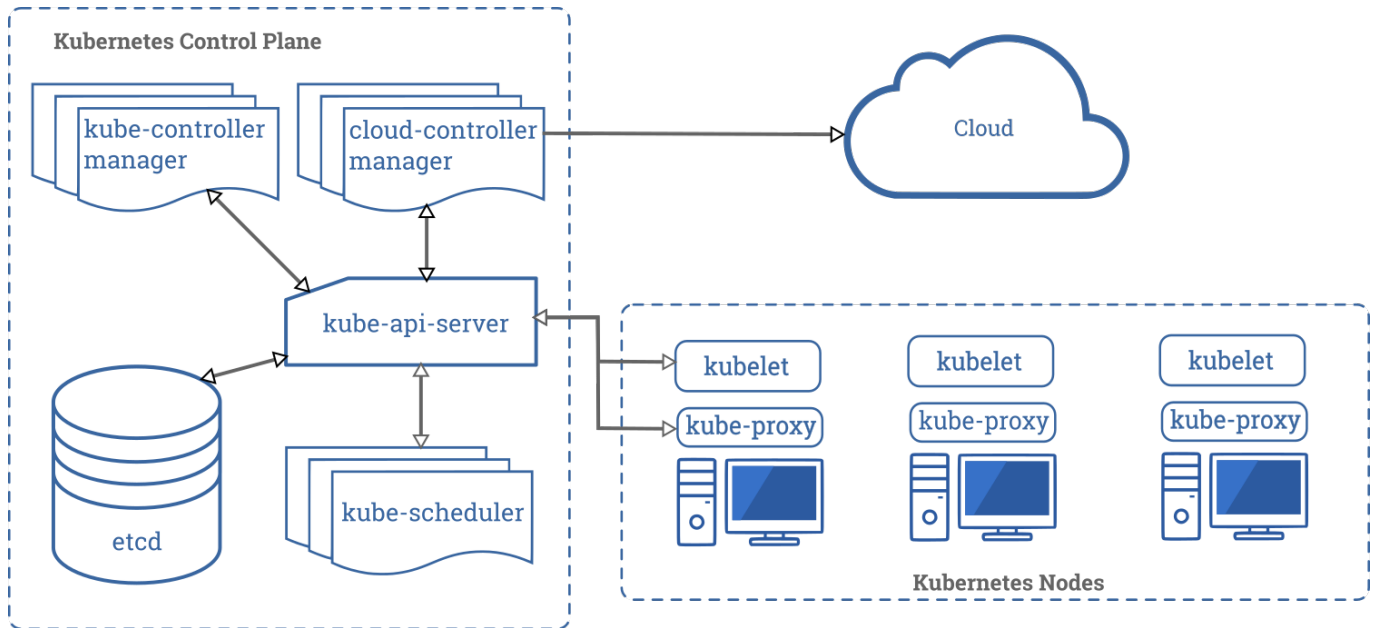
Figure 3.5 – Components of Kubernetes
Source: *https://kubernetes.io/docs/concepts/overview/components/*

3.3 General architecture of Batkube and its integration with Kubernetes and Batsim

3.3.1 Integration with Kubernetes

In order to adapt Kubernetes schedulers for use with Batsim we need to position ourselves between the Kubernetes scheduler and the cluster. There are several options here, at different levels of the cluster. All these are made possible by the fact that Kubernetes is entirely open source and can be reverse engineered and modified to suit our needs. First we present the options we considered but did not choose, and then we explain how we wrote a custom REST API which acts as a Kubernetes cluster to the scheduler, and as an event based scheduler to Batsim.

In between the api and the kubelets

This is the lowest level option. We position the simulator so as to simulate just the infrastructure and avoid tampering with Kubernetes resource management, which is done in their API. This approach would allow us to effortlessly use any Kubernetes scheduler once their API is supported by Batkube, and potentially produce the most accurate results. However, interactions between the kubelets and the API are not documented because the typical user is not supposed to have to deal with this part of Kubernetes. This would hinder the development of Batkube because a reverse engineering process would be required beforehand to understand the intricacies of internal Kuberenetes exchanges.

Custom client-go

Most Kubernetes schedulers rely on client-go[8], which is a Go client for the api-server. It is a library implementing various tools to help schedulers converse with the API. By altering this client and patching schedulers so they use our client instead, we can make it exchange with Batsim instead of the API.

Contrary to the kubelets, client-go is a user interface and therefore it is documented, facilitating reverse engineering of its source code. Still, it represents thousands of lines of code and altering it to our needs

---

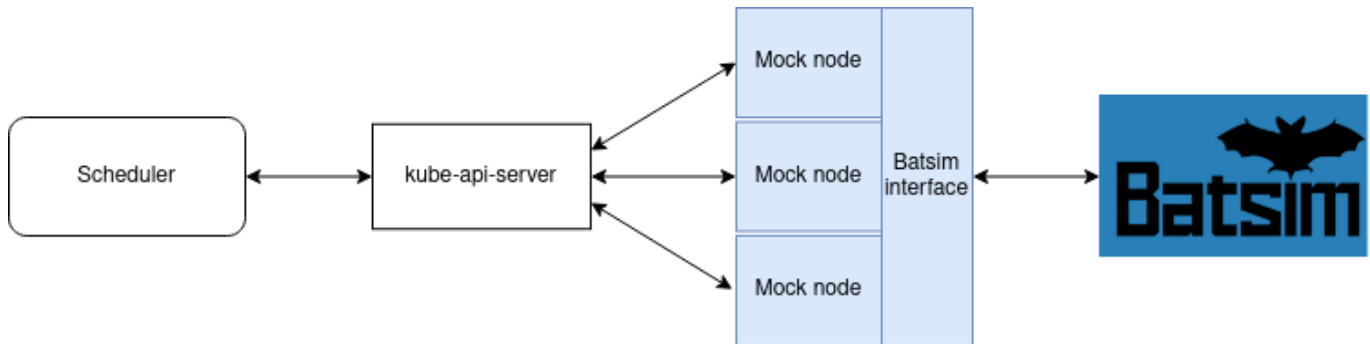[8]https://github.com/kubernetes/client-go
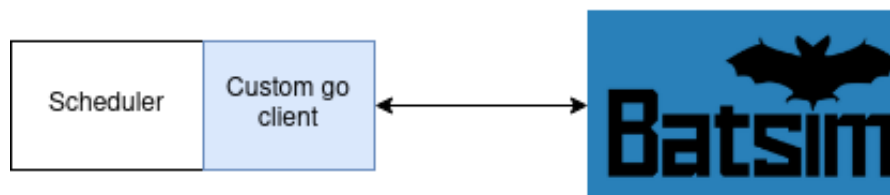
Figure 3.6 – Mocking the cluster itself.

Figure 3.7 – Custom Go client to redirect scheduler communications to Batsim

would not be an easy feat. The other drawback to this approach is that Batkube would only support schedulers written in Go and making use of client-go, although this should not be an issue as the only kubernetes scheduler we could find that does not rely on client-go is a toy scheduler written in bash [13].

Custom API

Re-implementing the API offers a middle ground between the low level and undocumented solution of the mock nodes, and the higher level and technically challenging solution of a client-go fork. Again, there are several options here.

A partial reimplementation of the API would save us the task of building a new API from the ground up. However this would imply digging deep into the api-server code in order to understand how the api is organized and what code we would have to alter. In the end, it is easier to simply build a new API, since there are tools to help us generate it from its specification.
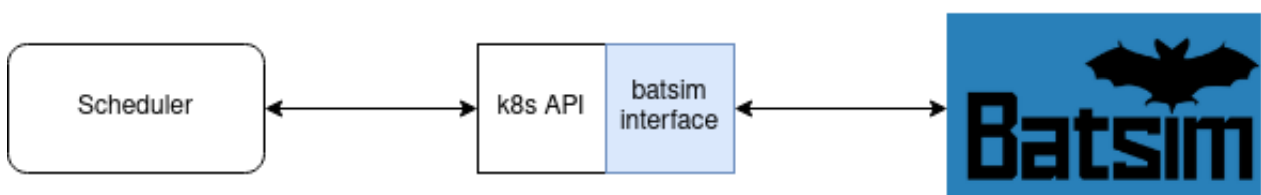

Figure 3.8 – Partial reimplementation of the api-server.

Furthermore, building a new API allows us to consider only the endpoints we need and have complete control over the source code. The technically challenging aspect here is Kubernetes resource management. Indeed, we need to provide the scheduler with expected informations about the cluster state if we want to obtain a correct behavior from it, and while the endpoints of the API are well documented, Kubernetes team did not write lengths about how resources are managed internally.

3.3.2 Architecture of Batkube

Figure 3.10 depicts the architecture of Batkube, which is written in Go. The central component is the **broker**. It handles the messages coming from Batsim and the scheduler while ensuring time synchronization
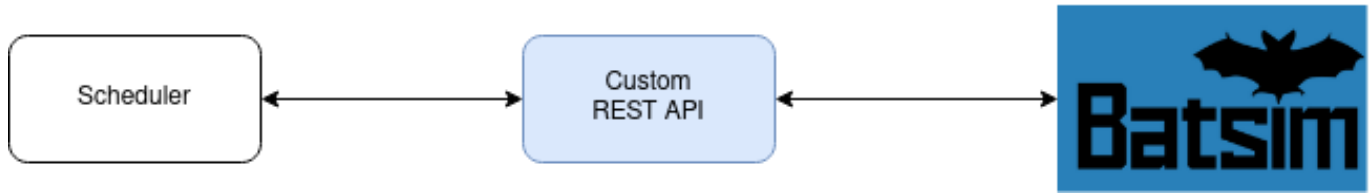
Figure 3.9 – Custom REST API in between the scheduler and Batsim.

between them. It is responsible for translating and forwarding messages between Batsim and the scheduler and orchestrates the synchronization between the two parties. **batsky-go** intercepts the calls to Go *time* library to ensure the scheduler's time is based on the simulation time instead of machine time. Time requests are forwarded to Batkube which replies with the current simulation time. The complete process to "hijack" scheduler time is explained in section 3.5. The **rest api** is the reimplementation of the Kubernetes api-server. It ensures the scheduler gets all the necessary information on the cluster state to make its scheduling decisions, and it is also the receiver of those decisions. Section 3.4 explains how we built this API and a tool to automatically generate its code from the api-server specification. **translate** is a utility package providing functions to translate Kubernetes resources to Batsim messages, and *vice versa*.
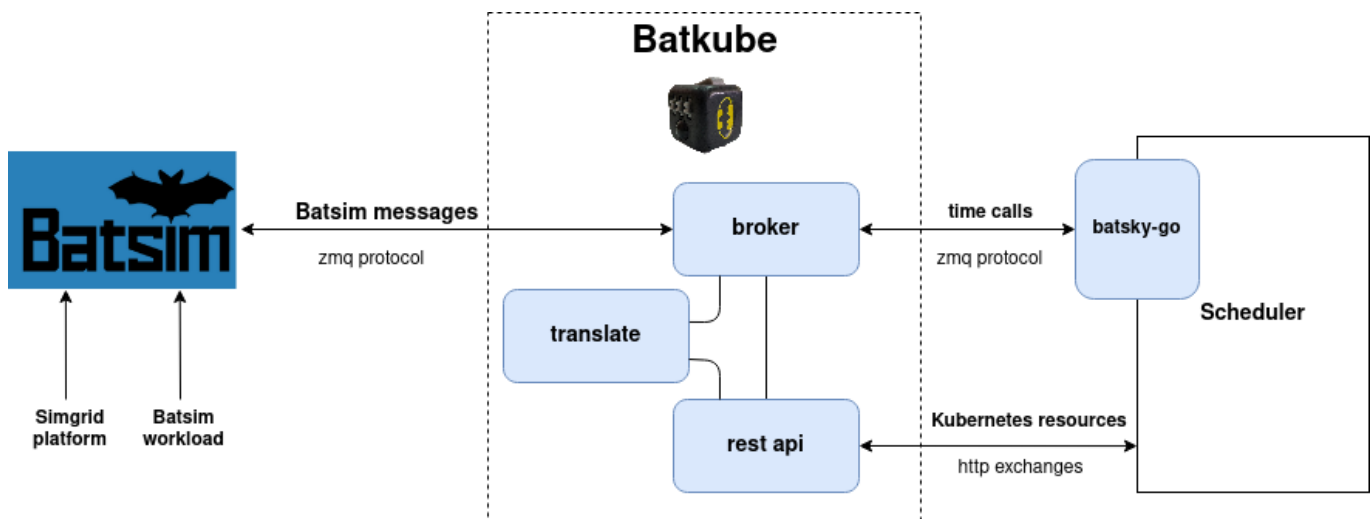

Figure 3.10 – Architecture of Batkube

3.4   Building the API

The API of Kubernetes follows the OpenAPI[9] 2 specification which is a standard for describing APIs. Luckily tools exist to generate such specification from source code, but also to generate code from a specification. Since the Kubernetes API specification is available on their reposotory[10], we were able to use such tools. This allowed us not to implement boiler plate code by hand and fill the gaps where they needed to be filled, leaving empty the endpoints we do not need. These endpoints can be dealt with later for future development of the simulator. For this project we used go-swagger[11] to generate our code and the API specification corresponds to the release 1.18 of Kubernetes. One downside of this method is that go-swagger forbids to tamper with the code of the server itself, although we did not need to during this project.

---

[9]https://www.openapis.org/about
[10]https://github.com/kubernetes/kubernetes/blob/release-1.18/api/openapi-spec/swagger.json
[11]https://github.com/go-swagger/go-swagger/tree/master/examples/stream-server

(a) Option A: patching the C library
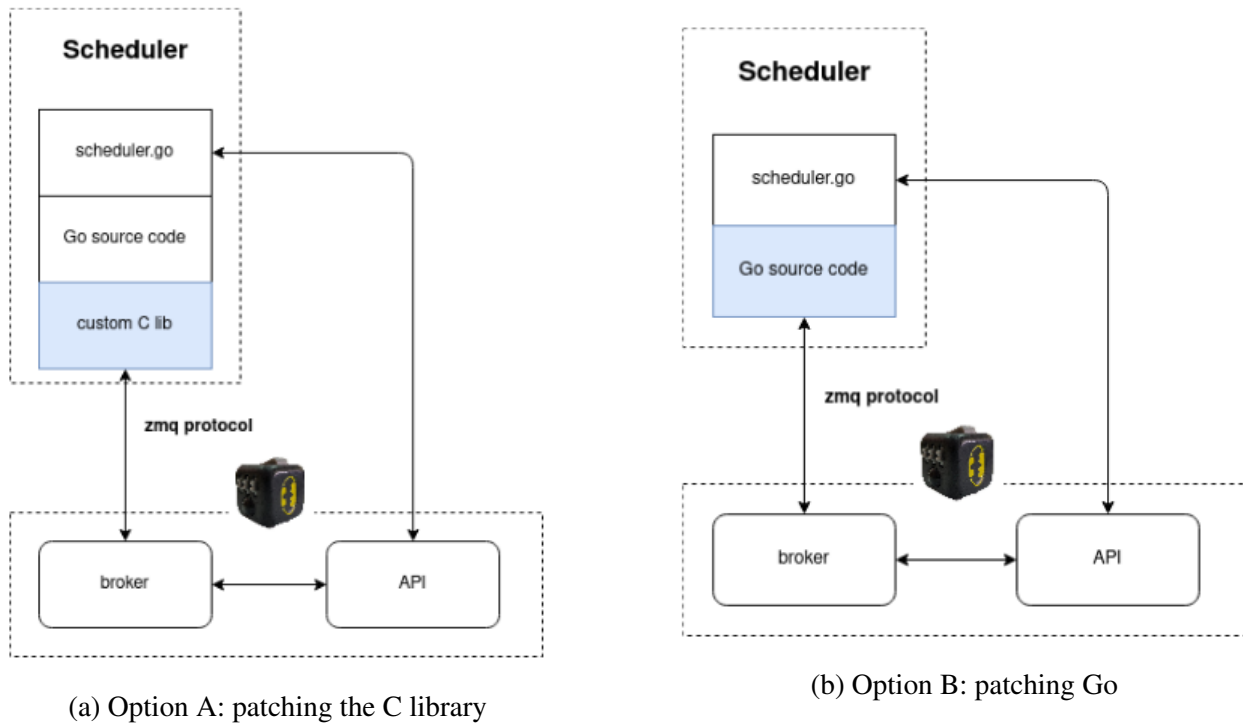
(b) Option B: patching Go

Figure 3.11 – Different approaches to intercept the scheduler calls to machine time

In order to enable communication between Batsim and the scheduler we need to translate Batsim messages into Kubernetes resources that can be retrieved by the scheduler and scheduler decisions to Batsim messages. Batsim Jobs are simply translated to pods. Jobs do exist in Kubernetes, but they are simply wrappers around pods: when submitting a job to the (real) api-server, the api ensures that a pod is created and executed to completion. In our case, we do not need such intermediate. Compute resources are simply translated to Kubernetes nodes.

3.5   Time interception

Kubernetes schedulers are not event based schedulers. They constantly check on the cluster state and make decisions accordingly, therefore they are based on machine time to make their decisions. However, in order to have correct simulations, the scheduler needs to be synchronized with simulation time. We then need to intercept all calls to machine time to redirect them to the simulator. There are two approaches to this: either we patch the C library used by Go (figure 3.11a) and compile Go using our library, or we patch Go's time library (**??**) and then patch the scheduler so it uses our library instead of the official one.

An attempt was first made for the custom C library, which is the lowest level solution. Going for the low level solution would truly redirect all calls to machine time which is something we can not guarantee with the second option, as we will see in section 3.5.2. This approach proved challenging due to circular dependency issues and was ultimately abandoned. We opted for the second option which consist of modifying Go source code, which requires some additional work to patch the schedulers but was actually easier to implement.

3.5.1   Redirection of time requests to Batkube

The module in charge of this time redirection is called **batsky-go**[12], which re implements `time.Now()` as well as timers and tickers. Timers are structures that are instantiated with a duration as input, that notify the

---

[12]https://github.com/oar-team/batsky-go

caller once this duration is elapsed. Timers can be reset, modified or deleted after initialization, which make their implementation tricky in a parallel computing context. Tickers are essentially the same structures as timers, except they regularly notify the caller with the given period of time instead of exiting like timers would.

In order to explain batsky-go algorithms as clearly as possible we need to provide some context about Go channels, which are structures made for running code in parallel. Go allows the user to run multi threaded code easily thanks to its **go routines**. Whenever in the code the user may create a routine which will create a new thread and launch the given code parallel to the encapsulating function. Essentially, this creates a child process – and introduces the risk of creating fork bombs[13]. Go channels are *"a typed conduit through which you can send and receive values with the channel operator, <-"*[14]

```
ch <- v     // Send v to channel ch.
v := <-ch   // Receive from ch, and
            // assign value to v.
```

Figure 3.12 – Usage of a Go channel

??

Algorithm 1: Time request (time.now())

   Result: Current simulation time
   Input: d: timer duration, req: requests channel, res: response channel map
   Output: now : simulation time

1 **if** *requester loop is not running* **then**
2   go runRequesterLoop() /* There can only 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

Algorithm 2: Requester loop

   Input: req: requests channel, res: response 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   */

---

[13]https://github.com/aaronryank/fork-bomb/blob/master/fork-bomb.go
[14]source: https://tour.golang.org/concurrency/2

3.5.2   Patching schedulers

TODO: Explain the AST approach and how it is not a plain sed. (conflict in the resources used, we have to replace all calls to time's **functions** and leave the **objects** as they are).

3.6   Time synchronization

TODO: explanatory text on this diagram. Explain the different parameters: timeout, max & base timestep & backoff multiplier, min delay, scheduler crash detection, fast forward on no pending jobs.
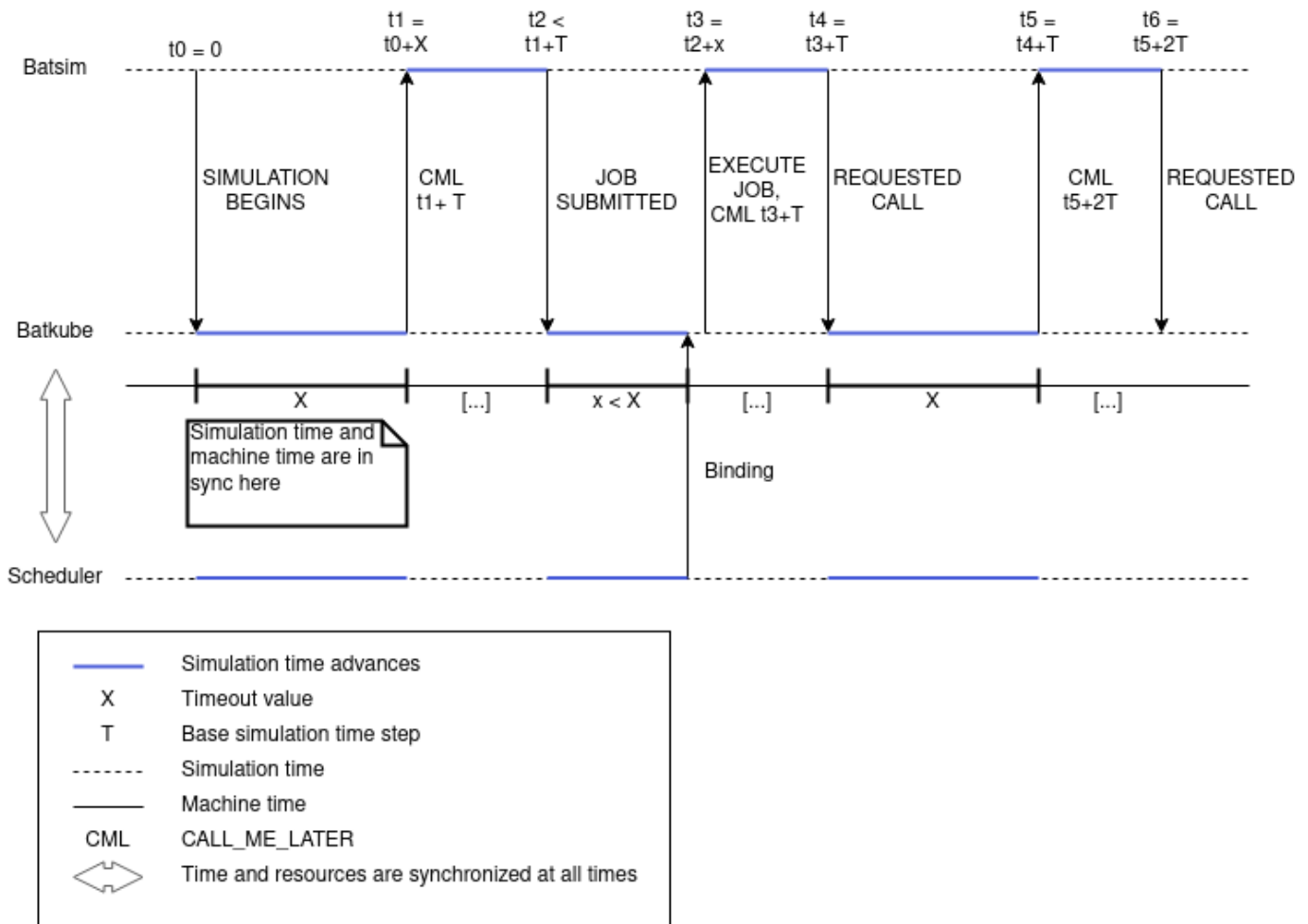

Figure 3.13 – Time sync between the three components. The broker has to take into account both machine time and simulation time.

— **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. All scripts used to run the experiment, process the workloads and generate the graphs present in this report – along with some results – are available on *batkube-test*[1] repository.

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.

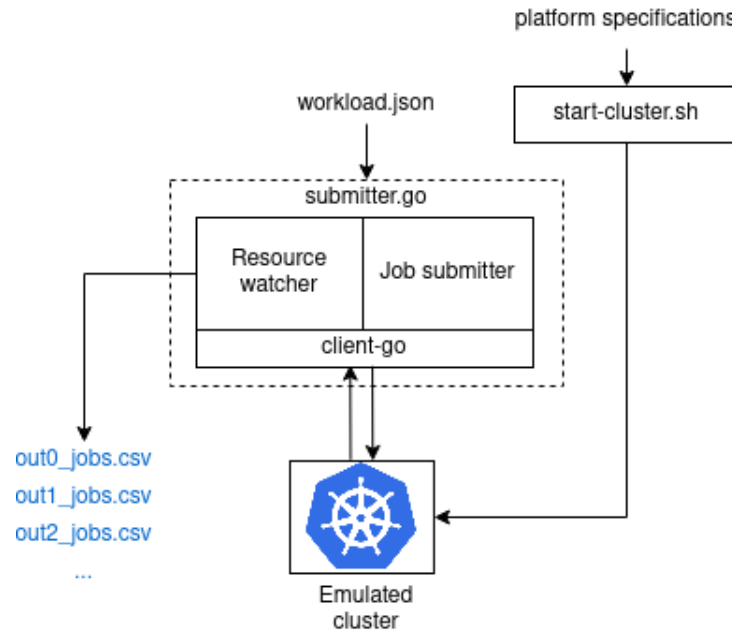
[1] github.com/oar-team/batkube-test

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

Batsim provides a tool to translate SWF files to its own json definition. It also works as a workload prepro-cessor, although we want to process SWF files very specifically to suit our needs which is why a custom script was implemented.

First, a trace in standard workload format (swf) was obtained on a web archive[2]. The chosen workload was `KIT-FH2-2016-1.swf` because it is the most recent and is relatively lightweight. Secondly, `evalys`[3] 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.

---

[2]https://www.cs.huji.ac.il/labs/parallel/workload/logs.html
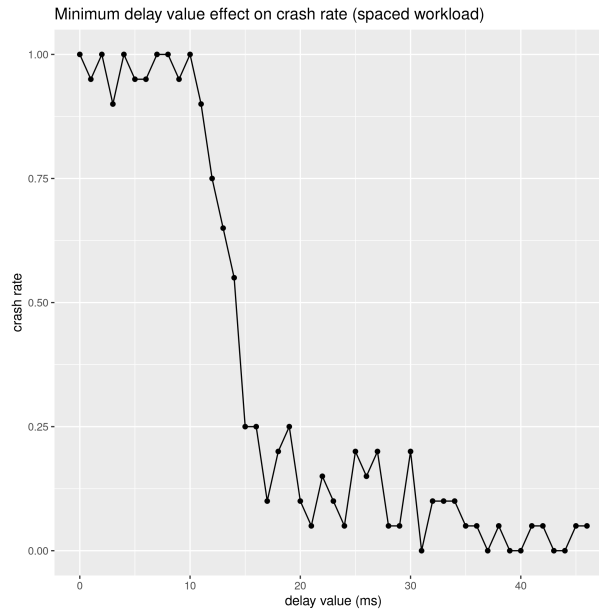[3]https://github.com/oar-team/evalys

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

- The *maximum simulation time step*, which is the maximum amount of time Batsim is allowed to jump forward in time.

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.
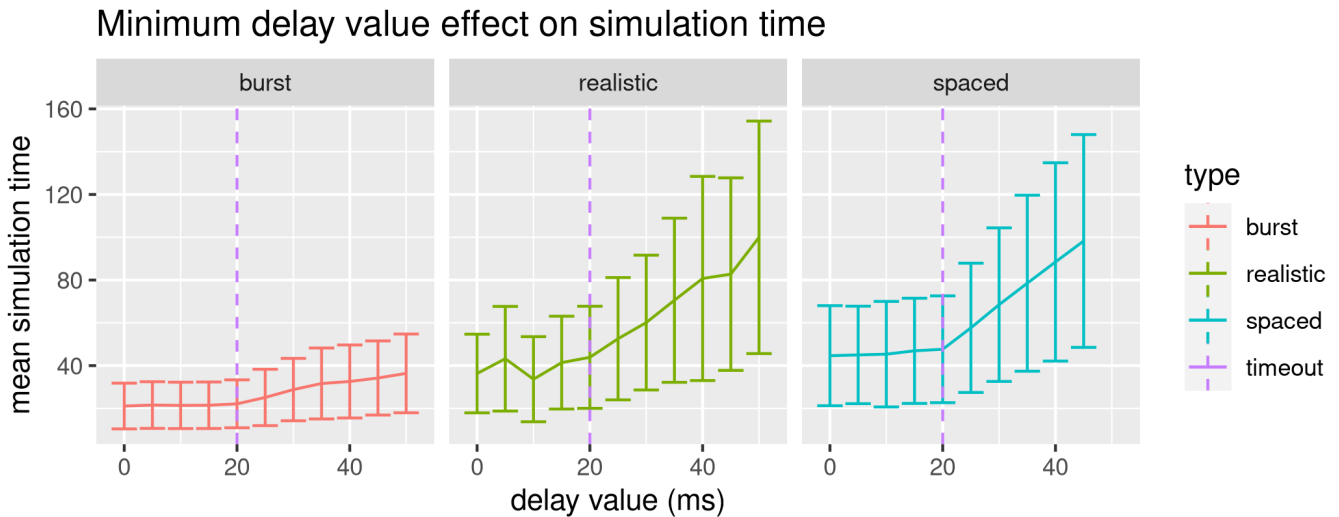
Minimum delay value effect on simulation time

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

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 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.
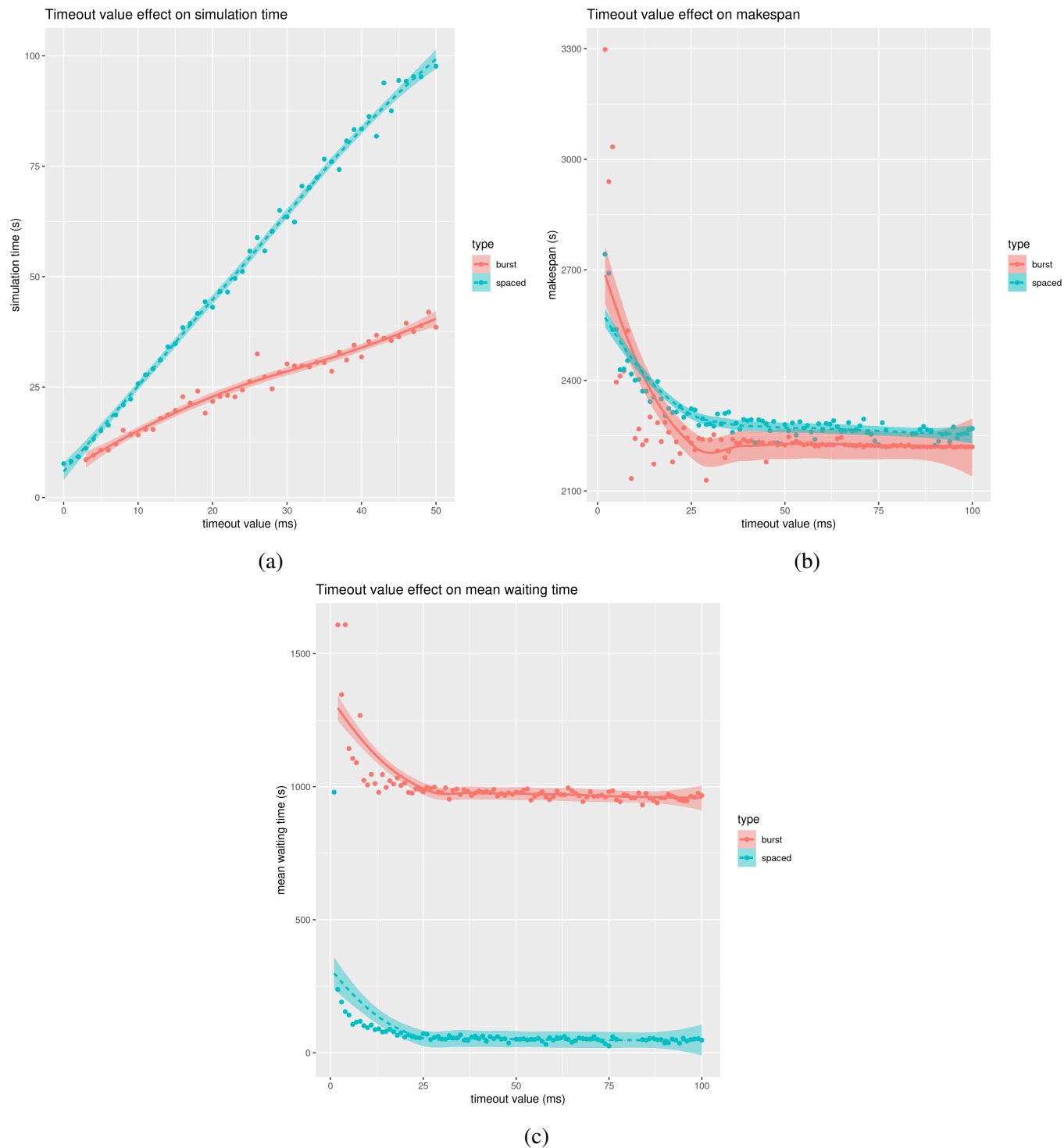
(a)

(b)

(c)

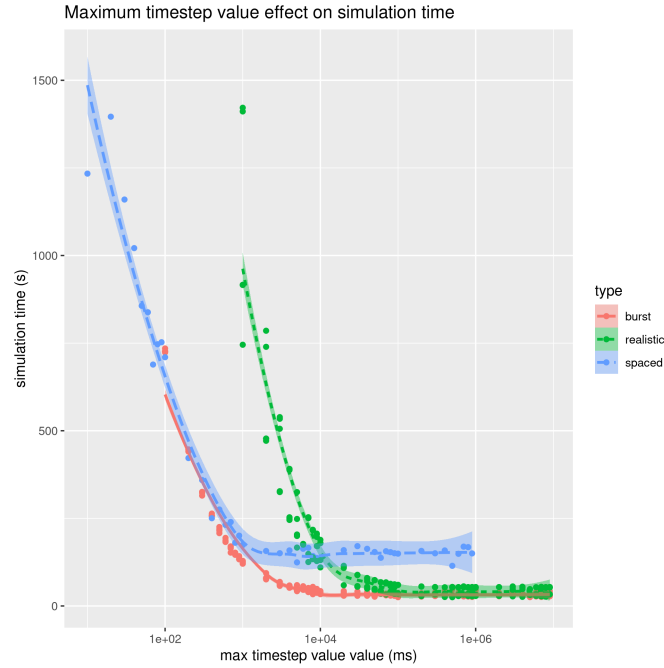Figure 4.4 – Effect of the timeout value on the simulation

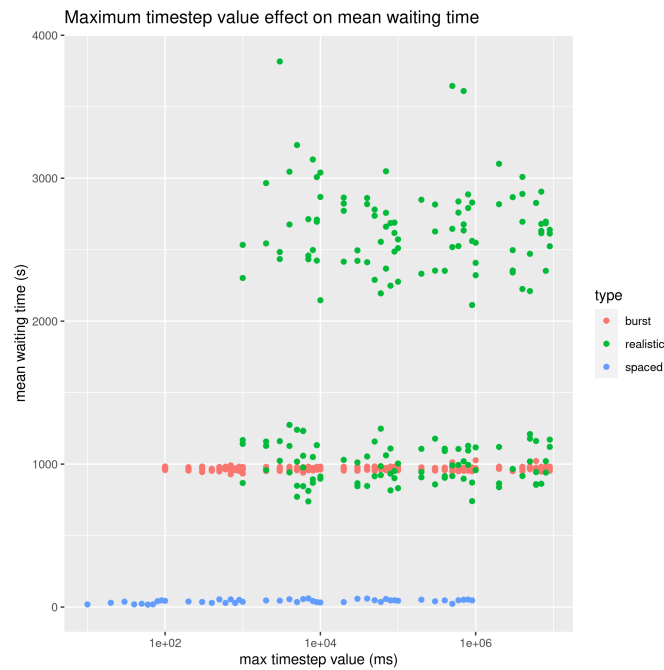Figure 4.5 – Effect of maximum timestep on simulation time

Figure 4.6 – Effect of maximum timestep on mean waiting time

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

To experiment with the maximum time step effect on the results we obtain, we run each workload with different values of `max-simulation-timestep` following a logarithmic scale. The other parameters are fixed to `min-delay=0s`, `timeout=50ms`. Also, the `base-simulation-timestep` was lowered to 10ms in order to test lower values of the maximum timestep (compared to the previous 100ms).

As expected, the simulation time decreases drastically when the maximum timestep increases. Still, this value reaches a minimum eventhough the maximum timestep keeps increasing. This happens because Batsim events are only so far appart in the simulation, and Batsim will always wake up before the maximum timestep is reached.

TODO: redo the experiment for the spaced wl and redo the graph for makespan (do not put all three wl on the same graph because the scale is so different).

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

Batkube is by no means scalable in comparison of existing batch schedulers (TODO: ref to some papers to prove this point). At its current state, it exists to prove adapting Kubernetes schedulers is possible, leaving optimization for future works. Also, Kubernetes is not optimized for batch scheduling and usage in the HPC field is still in its early states.

— 5 —
Future work

TODO: Project into the future. What needs to be fixed, and what needs to be implemented for Batkube to become a fully-fledged Kubernetes simulator based on the sound models of SimGrid.

Appendices

.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.

TODO

Limits: 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 for the scheduler (over allocating resources) (we talk about this in the evaluation part).

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] 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.

[4] J. Blazewicz, J.K. Lenstra, and A.H.G.Rinnooy Kan. "Scheduling subject to resource constraints: classification and complexity". In: *Discrete Applied Mathematics* 5.1 (1983), pp. 11 –24. ISSN: 0166-218X. DOI: https://doi.org/10.1016/0166-218X(83)90012-4. URL: http://www.sciencedirect.com/science/article/pii/0166218X83900124.

[5] Rajkumar Buyya and Manzur Murshed. "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing". In: *Concurrency and Computation: Practice and Experience* 14 (Nov. 2002). DOI: 10.1002/cpe.710.

[6] Rodrigo Calheiros et al. "CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services". In: (Apr. 2009).

[7] Y. Caniou and J. Gay. "Simbatch: An API for Simulating and Predicting the Performance of Parallel Resources Managed by Batch Systems". In: Apr. 2009, pp. 223–234. DOI: 10.1007/978-3-642-00955-6_27.

[8] Henri Casanova et al. "Developing Accurate and Scalable Simulators of Production Workflow Management Systems with WRENCH". In: *Future Generation Computer Systems* 112 (2020), pp. 162–175. DOI: 10.1016/j.future.2020.05.030.

[9] 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.

[10] Bruno Donassolo et al. "Fast and Scalable Simulation of Volunteer Computing Systems using Simgrid". In: June 2010, pp. 605–612. DOI: 10.1145/1851476.1851565.

[11] 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.

[12] Cristian Galleguillos, Zeynep Kiziltan, and Alessio Netti. "AccaSim: An HPC Simulator for Workload Management". In: *High Performance Computing*. Ed. by Esteban Mocskos and Sergio Nesmachnow. Cham: Springer International Publishing, 2018, pp. 169–184. ISBN: 978-3-319-73353-1.

[13] Justin Garrison. *bashScheduler*. URL: https://github.com/rothgar/bashScheduler (visited on 08/18/2020).

[14] 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.

[15] Dalibor Klusáček and Hana Rudová. "Alea 2 - Job scheduling simulator". In: *SIMUTools 2010 - 3rd International ICST Conference on Simulation Tools and Techniques* (Jan. 2010), p. 61. DOI: 10.4108/ICST.SIMUTOOLS2010.8722.

[16] Derrick Kondo. *SimBOINC: A simulator for desktop grids and volunteer computing systems*. 2007.

[17] *kube-batch*. URL: https://github.com/kubernetes-sigs/kube-batch (visited on 08/18/2020).

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

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

[20] *OAR next generation*. URL: http://oar.imag.fr/start (visited on 08/18/2020).

[21] 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).

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

[23] 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 - New York, NY, USA, 2017, 9–10. ISBN: 9781450351690. DOI: 10.1145/3154842.3154845. URL: https://doi.org/10.1145/3154842.3154845.

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

[25] 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.

[26] Pedro Velho et al. "On the Validity of Flow-Level Tcp Network Models for Grid and Cloud Simulations". In: *ACM Trans. Model. Comput. Simul.* 23.4 (Dec. 2013). ISSN: 1049-3301. DOI: 10.1145/2517448. URL: https://doi.org/10.1145/2517448.

[27] Hidehito Yabuuchi. *k8s-cluster-simulator*. URL: https://github.com/pfnet-research/k8s-cluster-simulator (visited on 08/16/2020).

[28] Yuan Chen. *A Toolkit for Simulating Kubernetes Scheduling at Scale*. URL: https://static.sched.com/hosted_files/kccncna19/97/KubeCon_JoySim.pdf. Nov. 2019.