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

TODO: maybe a summary of this section. (1 - kubernetes put in context, 2 - focus on the scheduling problem, 3 - studying computer infra in general, 4 - Batsim is a tool very well suited for answering the scheduling problem)

2.1 Cloud Computing

TODO: small intro on this part to clarify that this is to put Kubernetes in context.

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.

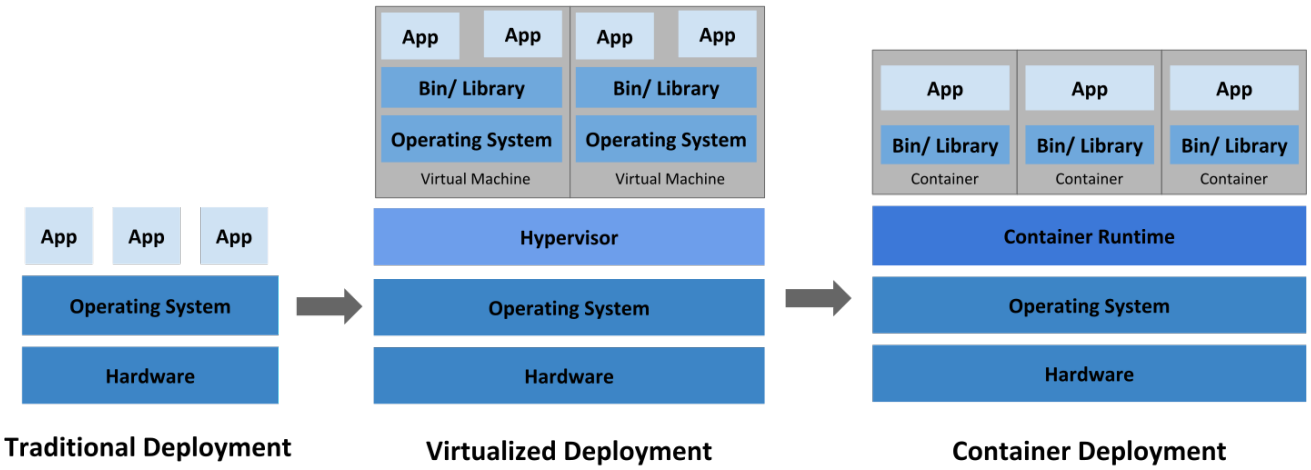

Figure 2.1: Evolution of application deployment.
Source: *https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/*

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.

TODO: outro to explain that eventhough this paradigm enabled developing new applications with ease, many questions remain: what infrastructure would be best suited for my application? how would different topologies and scheduling policies would affect performances? First, let's talk about the scheduling problem (in general) because it is the focus of experimentation with Batsim, and then discuss the general problem of the study of computing infrastructures.

2.2 Studying computer infrastructures

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

TODO : intro on the next sections

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

[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/

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.

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.

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

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 —
Implementation

3.1 Batsim

3.2 Kubernetes concepts

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

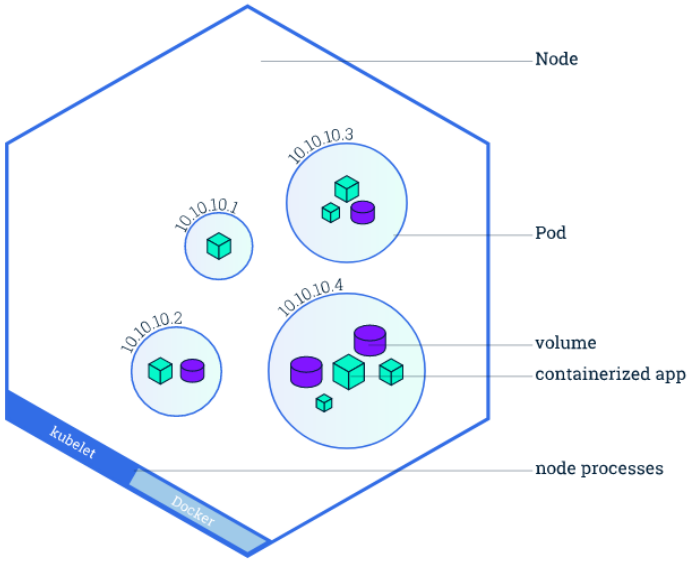

Figure 3.1: Node overview
Source: *https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/*

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

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

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

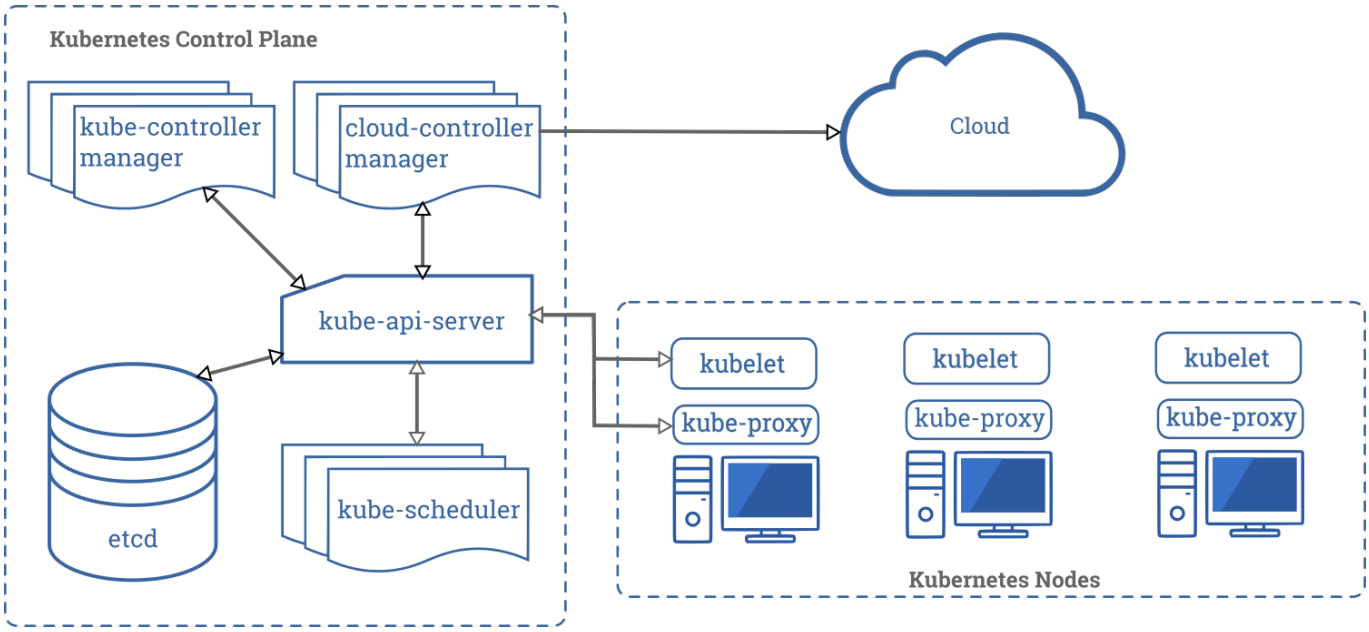

Figure 3.2: Components of Kubernetes
Source: *https://kubernetes.io/docs/concepts/overview/components/*

3.3 Objectives

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

3.4 Technical challenges

3.4.1 Translation

3.4.2 Time hijack

TODO

Algorithm 1: Requester loop

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

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

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

3.4.3 Time synchronisation

Algorithm 3: Synchronisation with timers

Input: timeout: time.Duration; simulationTimestep: time.Duration

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

Algorithm 4: Synchronisation with incremental time increases

Input: simulationTimeout: time.Duration; incrementTimeStep: time.Duration; incrementValue: time.Duration

```
1  stopWaitingForMessages := false
2  incremented := 0
3  while !stopWaitingForMessages do
4      getSchedulerDecisions()
5      if decisionReceived() or incremented >= simulationTimeout then
6          stopWaitingForMessages = true
7      else if timeSinceLastIncrement() > incrementTimeStep then
           /* this last condition is here to slow down this process and give time
              for the scheduler to take its decisions                        */
8          now = now + incrementValue
9          incremented = incremented + incrementValue
10 addCallMeLater() /* Add a CALL_ME_LATER event with timestamp now +
       incrementValue                                                        */
```

3.4.4 Re-building the API

— 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 want 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 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.

In these next sections, we study the simulator results against three parameters and across three workloads using the default kubernetes scheduler. Note that any other scheduler can be used, implying some work is done on the API to support their features.

The workloads are defined as follow:

- A *basic* workload, where 200 delays of value 160 are submitted at time zero.
- A *spaced* workload, where 200 delays of value 160 are submitted every 10 seconds.
- A *realistic* workload, which is extracted from a workload from a real system.

4.1 Testing against an emulated cluster

Protocol

TODO: validation against k3s, scripts to create a cluster with k3s, scripts to process swf files, evalys. Include illustration of the protocol.

Define *accuracy*

Limits

Results

4.2 Study of the simulator parameters

The objective here is to fine tune the simulator parameters in order to find a compromise between accuracy and scalability.

The parameters are:

- The *timeout* value when waiting for scheduler decisions. Default value: 20ms.

- The *minimum delay* we have to spend waiting for the scheduler. Default value: 10ms.

- The *maximum simulation time step*, which is the maximum amount of time Batsim is allowed to jump forward in time. Default value: 50s.

We first study these parameters one by one by fixing the other parameters to their default 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.

Timeout

Decreasing this value leaves less time for the scheduler to react: for example, if the scheduler needs 30ms to make a decision upon reception of an update in the resources state, 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). Intuitively, this will increase the simulation speed while leaving gaps in the decision process, causing a decrease in accuracy.

TODO: graph - timeout vs accuracy vs simulation time

Minimum delay

We observed that not leaving enough time to the scheduler each cycle causes it to put itself in a deadlock state. From there, no decision is made. The `-detect-scheduler-deadlock` option allows to exit the simulation with code 1, to effectively detect such crashes. We expect to see a very high percentage of crashes when this value is low, that will decrease until there is enough time for the scheduler to make its computations correctly.

TODO: graph - delay vs crash rate

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.

TODO: graph - time step vs acc vs sim time

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.

TODO: Facet graphs

Scalability experiments

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.

TODO: recreate large workloads on large systems

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].