Investigating Placement Challenges in Edge Infrastructures through a Common Simulator

Adwait Bauskar, Anderson da Silva, Adrien Lebre, Clement Mommessin, Pierre Neyron, Yanik Ngoko, Yoann Ricordel, Denis Trystram, Alexandre van Kempen

Investigating Placement Challenges in Edge Infrastructures through a Common Simulator

Anderson Da Silva[1], Clément Mommessin[1], Pierre Neyron[1], Denis Trystram[1], Adwait Bauskar[2], Adrien Lebre[2], Alexandre Van Kempen[2], Yanik Ngoko[3], and Yoann Ricordel[3]

[1]Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, Grenoble, France, {firstname.lastname}@inria.fr
[2]Inria, LS2N, UMR 6004, IMP Atlantique, Nantes, France {firstname.lastname}@inria.fr
[3]Qarnot Computing, Montrouge, France {firstname.lastname}@qarnot-computing.com

Abstract

Efficiently scheduling computational jobs with data-sets dependencies is one of the most important challenges of fog/edge computing infrastructures. Although several strategies have been proposed, they have been evaluated through ad-hoc simulator extensions that are, when available, usually not maintained. This is a critical problem because it prevents researchers to –easily– conduct fair evaluations to compare each proposal.

In this short paper, we propose to address this limitation by presenting the first elements of a common simulator. More precisely, we describe an on-going project involving academics and a high-tech company that aims at delivering a dedicated tool to evaluate scheduling policies in edge computing infrastructures. This tool enables the community to simulate various policies and to easily customize researchers/engineers' use-cases, adding new functionalities if needed. The implementation has been built upon the Batsim/SimGrid toolkit, which has been designed to evaluate batch scheduling strategies in various distributed infrastructures. Although the complete validation of the simulation toolkit is still on-going, we demonstrate its relevance by studying different scheduling strategies on top of a simulated version of the Qarnot Computing platform, a production edge infrastructure based on smart heaters.

1 Introduction

The proliferation of Internet of Things (IoT) applications [7], as well as the advent of new technologies such as Mobile Edge computing [3], and Network

Function Virtualization [17] (NFV) have been accelerating the deployment of Cloud Computing like capabilities at the edge of the Internet. Referred to as the fog [8] or the edge computing [20] paradigms, the main objective is to perform on demand computations close to the place where the data are produced and analyzed in order to mitigate data exchanges and to avoid too high latency penalties [26]. Among the open questions our community should address to favor the adoption of such infrastructures is the computation/data placement problem i.e., *where to transfer data-sets according to their sources and schedule computations to satisfy specific criteria*. Although several works have been dealing with this question [4, 9, 11, 18, 21, 23, 24], it is difficult to understand how each proposal behaves in a different context and with respect to different objectives (scalability, reactivity, etc.). In addition to having been designed for specific use-cases, they have been evaluated either using *ad hoc* simulators or in limited in-vivo (i.e., real-world) experiments. These methods are not accurate and not representative enough to, first, ensure their correctness on real platforms and, second, perform fair comparisons between them.

Similarly to what has been proposed for the Cloud Computing paradigm [15], we claim that a dedicated simulator toolkit to help researchers investigate fog/edge scheduling strategies should be released soon. However, we claim that using placement simulators for Cloud Computing is not appropriate to study fog/edge challenges. In addition to resource heterogeneity, network specifics (latency, throughput), and workloads, fog/edge computing infrastructures differ from Cloud Computing platforms because of the uncertainties: connectivity between resources is intermittent and storage/computation resources can join or leave the infrastructure at any time, for an unpredictable duration. In other words, a part of the infrastructure can be isolated for minutes/hours preventing accessing some data-sets or assigning new computations.

In this preliminary work, we present several extensions we implemented on top of the Batsim/SimGrid toolkit [10, 12] to favor fair evaluations and comparisons between distinct scheduling strategies for fog/edge infrastructures. In particular, we developed an external module to inject any type of event that could occur during the simulation (e.g., a machine became unavailable at time t). We also implemented a Storage Controller, to supervise all transfers of data-sets within the simulated platform.

We chose to extend Batsim/SimGrid instead of available fog/edge simulators [13, 22, 25] for several reasons:

- Batsim has been especially designed to test and compare batch scheduling policies in distributed infrastructures. In other words, the design of Batsim enforces researchers to use the same abstractions and thus, favor straightforward comparisons of different strategies, even if they have been implemented by different research groups.

- The accuracy of the internal models (computation and network) of SimGrid has been already validated.

- Batsim provides a Python API that makes the development of a scheduling

strategy simple.

By extending Batsim to the fog/edge paradigm, we target a tool that will enable researchers/engineers to re-evaluate major load balancing strategies of the state-of-the-art. In particular, we expect researchers to study whether scheduling algorithms that have been proposed two decades ago in desktop computing platforms, volunteer computing and computational grids [5, 6] can be slightly revised to cope with edge specifics. While edge workloads differ from best-effort jobs, desktop/volunteer computing and computational grids have several characteristics that are common to fog/edge platforms.

Although the validation of our extensions as well as the integration of representative edge workloads is still on-going, the first building blocks we implemented enabled us to study an edge infrastructure as complex as the *Qarnot Computing* platform [1]. The *Qarnot Computing* infrastructure is a production platform composed of 3,000 diskless machines distributed across several locations in France and Europe. Each computing resource can be used remotely as traditional cloud computing capabilities or locally in order to satisfy data processing requirements of IoT devices that have been deployed in the vicinity of the computing resource. As such, the *Qarnot* platform is a good example of an edge infrastructure, with computing units and mixed local/global job submissions with datasets dependencies.

After giving an overview of our edge placement simulator, we present how the *Qarnot* infrastructure has been instantiated on top of our framework, how we used our injector to simulate the *Qarnot* workload and, finally, how we evaluated different scheduling strategies for job placement and data movements.

The rest of the paper is structured as follows. Section 2 gives an overview of the Bastim/SimGrid toolkit and the extensions we implemented. Section 3 presents the *Qarnot Computing* use-case. Section 4 describes how we simulated this case study and Section 5 discusses a first analysis of different scheduling strategies for the *Qarnot* platform. Section 6 presents the related work. Conclusion and future works are given in Section 7.

2 A dedicated Scheduling Simulator for edge platforms

At coarse-grained, our proposal relies on a few extensions developed on top of the Batsim/SimGrid toolkit. Released in 2017, Batsim [12] delivers a high-level API to facilitate the development of batch scheduling algorithms that can be then simulated on top of SimGrid [10], the well-proven simulator toolkit for distributed infrastructures. Thus, we rely on high-level tools that have been proposed and already validated. We have customized some parts to reflect the edge specifics, and propose some extensions as explained in the followings.

2.1 Operational components

We discuss in this section the role of the different components, namely SimGrid, Batsim, the *decision process* and their interactions.

2.1.1 SimGrid

SimGrid [10] is a generic simulator framework that enables simulation of any distributed system. In addition to providing the program to be evaluated, performing simulations with SimGrid requires (i) writing a platform specification, (ii) formatting workload input data, (iii) interfacing the program to evaluate. The choice of using SimGrid as the main engine for Batsim is mainly due to its relevance in terms of performance, as well as its validity that has been backed-up by many publications [2]. Moreover, it enables the description of complex infrastructures, such as hierarchical ones, that are composed of many interconnected devices with possibly highly heterogeneous profiles. Finally, the injection of external events on demand, such as node apparitions/removals or network disconnections, has allowed us to easily simulate complex systems such as fog/edge infrastructures.

2.1.2 Batsim and the decision process

Batsim [12] is an infrastructure simulator for jobs and I/O scheduling, built on top of SimGrid, to help the design and analysis of batch schedulers. Batch schedulers, *a.k.a.,* Resource and Jobs Management Systems, are systems in charge of managing resources in large-scale computing centres, notably by scheduling and placing jobs. Batsim allows researchers to simulate the behaviour of a computational platform on which a workload is executed according to the rules of a decision process. It uses a simple event-based communication interface; as soon as an event occurs, Batsim stops the simulation and reports what happened to the *decision process*.

The decision process embeds the actual scheduling code to be evaluated. In other words, in order to simulate a given scheduling algorithm, an experimenter has to implement this decision process. Comparing different algorithms consists in switching between different decision processes, which is straightforward. Internally, the decision process reacts to the simulation events received from Batsim, takes decisions according to the given scheduling algorithm, and drives the simulated platform by sending back its decisions to Batsim. Batsim and the decision process communicate via a language-agnostic synchronous protocol. In this work, we used Batsim's Python API to implement our decision process, which provides functions to ease the communication with Batsim.

For more details on Batsim and SimGrid mechanisms, we invite the reader to refer to Chapter 4 of Millian Poquet's manuscript [19].

2.2 Extensions

We are working on a couple of extensions to deal with edge challenges. In this section, we present the ones that are already available, namely a SimGrid plug-in, the events injector and the storage controller. Modifications made in Batsim[1] and its Python API[2] for this work are available in a separate branch of their main repository.

2.2.1 Batsim/SimGrid plug-in

One of the strengths of SimGrid is its plug-in architecture. For instance, one plug-in of interest, that have been validated by a previous work of the SimGrid team, is the estimation of the energy consumption of a host for a period of time, given inputs such as the power state of a host, the number of computing cores and the load of each core [14].

In the context of edge IoT platforms, providing other metrics to the simulation is of interest, and can take the form of new plug-ins. We extended the Batsim communication protocol to benefit from those plug-ins information (i.e., information related to SimGrid plug-ins can be reified on demand to the decision process).

As we will discuss later in the document, we leveraged this extension in the *Qarnot* case study. Concretely, we developed an additional service on top of the aforementioned energy plug-in that computes the temperature of a host from its energy consumption and other physical parameters.

2.2.2 External events injector

To simulate the execution of a fog/edge infrastructure, which by essence is subject to very frequent unexpected or unpredictable changes, our simulator offers the opportunity to inject external events on demand. Those events impacts the behaviour of the platform during the simulation and thus the choices of the scheduling strategy. For example, one would be interested in studying the behavior and resilience of a scheduling policy when a range of machines may become unexpectedly unavailable for a period of time, due to a failure or action occurring at the edge (e.g., from a local user).

The mechanism we implemented replays external events that occurred at a given time. When an event occurs it is handled by the main process of Batsim, that updates the state of the platform and the simulation, and then forwarded to the decision process.

We implemented the events to be the most generic possible. An event is represented as a JSON object that contains two mandatory fields: a *timestamp*, that indicates when the event should occur, and *type*, the type of the event. Then, depending on the type of event, other fields can complement the event description, such as the name of the unavailable resource for example, the new

[1] https://gitlab.inria.fr/batsim/batsim/tree/temperature
[2] https://gitlab.inria.fr/batsim/pybatsim/tree/temperature

value of an environment parameter such as the network bandwidth, or anything that is of interest to the decision process. External events are injected in Batsim by one of its internal processes, which reads the list of events from an input file containing one of the above described JSON objects per line.

This event injection mechanism is generic by concept: users can define their own types of events and associated fields, which will be forwarded to the decision process without any modification in the code of Batsim.

2.2.3 Storage Controller

The Storage Controller is a Python module that exposes multiple functions to the scheduler in order to manage the storage entities as well as the data transfers. In order to give the scheduler reliable information, it keeps track in real-time of the platform status, i.e., the on-going data transfers, the available resources, etc. It also manages all aspects related to caching policies, while offering advanced features such as speculation.

3 Case study: the Qarnot Computing platform

We present in this section the platform of the *Qarnot Computing* company, which serves as an example for our case study.

3.1 Infrastructure overview

Qarnot Computing has been incorporated in 2010 to develop a disruptive solution able to turn IT waste heat into a viable heating solution for buildings. The infrastructure is distributed in housing buildings, offices and warehouses across several geographical areas in France and Europe, in each situation valorizing the waste heat produced by IT computations to heat air and water for the building. As of writing this paper, the whole platform is composed of about 1,000 computing devices (*QRads*) hosting about 3,000 diskless machines, and is growing quickly. The diskless machines have access to some storage area present on the deployment site (*QBox*), shared as NFS through a LAN. In a typical configuration a computing machine has a 1 Gbps uplink to a common switch, which then has up to 40 Gbps uplink to the *QBox*. The latency between a computing machine and its storage area is of the order of 1 ms. The various deployment sites are connected to the Internet using either a public or enterprise ISP, with characteristics varying from 100 Mbps to 1 Gbps symetric bandwidth to the Internet, and about 10 ms latency to French data centers used by *Qarnot* to host control and monitoring infrastructure, central storage services, and gateways to its distributed infrastructure.

On a daily basis, from a few hundred to several thousands of batch jobs are processed by *Qarnot* batch computing PaaS, and thousands of cores are provisioned for corporate customers deploying infrastructure. Up to tens of GBs of data are replicated from central storage to edge computing sites.

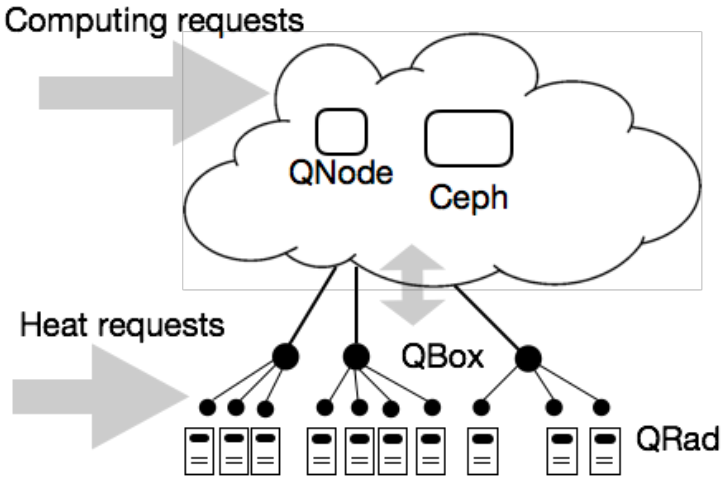

Figure 1: Scheme of the Qarnot platform.

Qarnot deploys high performance computing hardware and storage capacity to buildings, which makes it a fitting infrastructure to locally gather and process data that is generated at the edge of the network (for instance by smart buildings). One objective of the edge simulator is to evaluate evolutions of the *Qarnot* architecture to handle such local use-cases. It will allow investigating the edge infrastructure dimensioning as well as the optimization of the local data and processes placement with regard to the global ones. This can reduce global data movements, enable buildings to be autonomous in terms of IT and to handle Internet connectivity loss gracefully.

3.2 Platform Terminology

The jobs and resource manager of the *Qarnot* platform, named *Q.ware*, is based on a hierarchy of 3 levels as shown in Figure 1: the *QNode*-, the *QBox*- and the *QRad*-level. The QNode is the root node, a "global" server that takes placement decisions for the whole platform. It can be viewed as a load balancer for the platform. Connected to this QNode there are the QBoxes, which are "local" servers in smart buildings that take scheduling decisions locally on their own computing nodes. Each QBox is in charge of a set of computing nodes, the QRads, which are composed of one or several computing units denoted by *QMobos*.

Moreover, a centralized storage server, the *CEPH*, is present at the QNode-level while each QBox has its own local storage disk. From a physical point of view, the QNode and CEPH are on the cloud while QBoxes are distributed over smart buildings of several cities. QRads among a building are distributed in different rooms.

The *Qarnot* platform receives two types of user requests: requests for com-

puting and requests for heating. The computing requests describe the workload to be executed on the platform. They are made by users that first upload input data needed to execute their jobs (named *QTasks*) to the centralized server and upload a Docker image either to the centralized server or the Docker Hub. Then, they submit the QTasks to the QNode. A QTask can be decomposed in a bag of several *instances* that share the same Docker image and data dependencies, but with different command argument. This can be used for example to process each frame of a given movie, with one frame or a range of frames per instance.

The heating requests are made by inhabitants that can turn on and off the smart heaters in their homes, or set a target temperature for rooms to be reached as soon as possible. Since the computing units in a smart heater are unavailable when cooling is necessary, and are available otherwise, such changes increases the heterogeneity challenges of and edge infrastructure: the computation resource does not simply appear or disappear but also vary according to the heating needs.

3.3 Current scheduling policy

Whenever QTasks are submitted on the platform all the data dependencies should be uploaded to the CEPH. To be executed, these QTasks have to be scheduled to the QBoxes and then scheduled onto QMobos through two scheduling steps.

The first step takes place at the QNode-level. The QNode greedily dispatches as much instances of the QTasks ordered by priority on QBoxes, depending on the amount of QMobos available for computation on each QBox.

The second step takes place at the QBox-level. Upon receiving instances of a QTask, the QBox will select and reserve a QMobo for each instance and fetch from the CEPH each missing data dependency before starting the instances.

Notice that, at all times, a *Frequency Regulator* runs on each QRad to ensure that the ambient air is close to the target temperature set by the inhabitant, by regulating the frequencies of the QMobos and completely turning off a QRad if it is too warm. Moreover, whenever there is no computation performed on the QMobos while heating is required, some "dummy" compute-intensive programs are executed to keep the QRad warm.

4 Simulated Platform

We present in this section how we modeled the *Qarnot* platform and explain how we instantiated the inputs that were required for the simulation.

4.1 Qarnot to Batsim/SimGrid Abstractions

Figure 2 depicts the real and the simulated platforms. Each QMobo of the platform is simulated as a SimGrid host as they are the only computing units of the platform. QMobos belonging to the same QRad are aggregated in the same

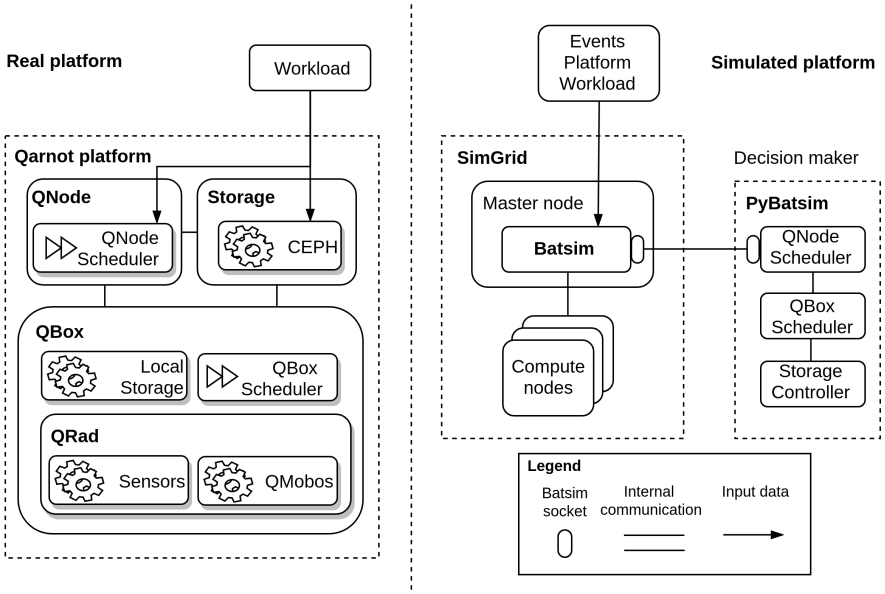

Figure 2: Comparison between the real and simulated *Qarnot* platform.

SimGrid zone, as well as QRads of a same QBox, and all QBoxes of the QNode. The management of storage spaces is done by adding special hosts which handle the *storage* role. Thus, in each QBox zone there is one additional storage host for the QBox disk. Similary, there is one storage host in the QNode zone to represent the CEPH. For the computing requests, each instance of a given QTask can run independently of the others, so we transcribed each instance as one Batsim job, with the same data-set dependencies and submission time for jobs belonging to the same QTask. Regarding the heating requests, each change of the target temperature of a QRad is simulated as an external event injected in the simulation, as well as when a QRad was turned off for being too warm. Finally, the schedulers of the QNode- and QBox-level (including the Frequency Regulator) were implemented in Python and both live in the same process, along with the Storage Controller.

4.2 Extracting Qarnot Traces

A log extractor was built to generate all the input files to feed Batsim and the decision process from real logs of the *Qarnot* platform, for a given time period. These files describe the platform, the workloads and their data-dependencies, the list of data-sets and all events that are mandatory to simulate the *Qarnot* system.

While we had to build a specific extractor to gather and format information from the *Qarnot* logs, generated files are generic with respect to Bat-

sim/SimGrid interfaces. In this sense, one could easily consider simulating another platform by taking our extractor as an example and adapt it to its needs.

4.2.1 Platform description

The definition of the platform is an XML file readable by SimGrid. The file describes the whole platform to simulate, with in details: A list of QBoxes with for each: the id, the network bandwidth and latency to the CEPH and to its QBoxes, the storage disk host and its size, the localization and a list of QRads. For each QRad: the id and a list of QMobos. For each QMobo represented by a SimGrid host: the id, the list of speeds and corresponding power usage, and the coefficients required for the temperature plug-in.

4.2.2 Workload description

The workload is represented by a JSON file containing a list of job descriptions and a list of profile descriptions. Job descriptions are defined by the user requests and contain: the id, the submission time, and the job profile to use. Profile descriptions represent how a job should be simulated, plus other specific information, and contain: the type of the job to simulate, the number of flops to compute, the job priority and the list of data-sets required as inputs.

The list of data-sets is also described as a list of JSON objects, with one per line in the file read by the *decision process* and fed to the Storage Controller. Each data-set object is represented by two fields: *id*, a unique identifier for the data-set, and *size*, the size in bytes of the data-set. This list of data-sets is read by the Storage Controller upon initialization of the schedulers and the storage representing the CEPH is filled accordingly.

4.2.3 External events description

We extracted the following events from the logs of the real platform:

- *qrad_set_target_temperature*: containing the id of the QRad and its new target temperature. This event informs the associated QBox that the temperature target of a QRad has changed.
- *machine_available* and *machine_unavailable*: containing a list of resources impacted by this event.
- *site_set_outside_temperature*: containing the location and its new outside temperature. This event is directly forwarded to the temperature plug-in(see Section 2.2.1).

As discussed in Section 2.2.2, each event is timestamped and is described as a JSON object.

We finally added a *stop_simulation* event to ask the scheduler to kill all executing jobs and reject waiting jobs to strictly stop the simulation after a given time.

5 Simulation results

In order to illustrate the versatility of our tool, we compare various scheduling strategies based on real-world traces of the *Qarnot* platform. We discuss in this section preliminary results.

Table 1: Metric values of the proposed policy variants compared to the *Qarnot* scheduler (Standard), for 1 week of simulation.

Metrics	Standard	LocalityBased	FullReplicate	Replicate10	Replicate3
Total transferred data (GB)	193.83 –	221.82 (+14.44%)	1110.27 (+472.81%)	668.96 (+245.13%)	414.19 (+113.69%)
Mean waiting time (s)	54.24 –	48.01 (−11.49%)	26.26 (−51.59%)	41.08 (−24.26%)	39.24 (−27.65%)
Mean bounded slowdown	2.57 –	2.65 (+3.11%)	1.39 (−45.91%)	1.85 (−28.02%)	2.13 (−17.12%)

5.1 Scheduling Policies

Along with the real *Qarnot* scheduler that serves as a baseline for our experiments (see Section 3.3), we also implemented four variants of the scheduling policy at the QNode-level.

First, the *LocalityBased* scheduler, that gives priority to the QBoxes already having in disk all data dependencies of the QTask to be dispatched. This first variant aims at avoiding useless data transfers if some QBoxes already have the required data dependencies of a given QTask.

The second scheduler, namely the *FullReplicate*, considers that all data dependencies of a QTask are replicated on all QBox disks before that QTask arrives in the system. This variant aims at visualizing the behaviours of the scheduling policy without any impact of the data movements.

Finally, the *Replicate3* and *Replicate10* schedulers, that respectively replicates all data dependencies of a QTask on the 3 and the 10 least loaded QBox disks upon submission of that QTask before applying the same dispatching policy as the LocalityBased one. These two schedulers are a trade-off between the two first variants. They aim at reducing the waiting time of the instances of QTasks by providing more QBox candidates for the LocalityBased dispatcher.

5.2 Results

Since the simulation and the scheduling algorithms are deterministic, we ran one simulation with each scheduler for various simulation inputs corresponding to logs of the *Qarnot* platform during 1 day, 3 days or 1 week. The running

time of one simulation was less than 5 minutes for a 1-day simulation, around 10 minutes for a 3-day simulation, and less than 35 minutes for a 1-week simulation, with about 15% of the time spent in the decision process.

We compared the different scheduling policies according to various metrics, including the total amount of transferred data, the mean waiting time of the jobs and the mean bounded slowdown. For a job, the bounded slowdown (also called *bounded stretch*) represents the ratio between the time spent in the system (including the waiting time) versus the maximum between 1 and the time, in seconds, to effectively process the job.

Due to lack of space, and as the simulations of the three periods (1-day, 3-days, 1-week) lead to similar conclusions, we only present in Table 1 the results for the 1-week period. This 1-week trace contains 5,506 instances grouped in 1,019 QTasks and uses 47 different data-sets.

The results show that, as expected, the three policies using replication improved the scheduling metrics compared to Standard, with a cost of an increasing size of the data transfers. Regarding the LocalityBased scheduler, the improvement of performance are quite low and even worse for the mean bounded slowdown, compared to the standard algorithm. This is quite surprising as, for example, the total transferred amount of data actually increased, while the initial purpose of this scheduler was to leverage the data transfer already performed. These unexpected results has led us to conduct additional investigations that shed light upon some interleaving of scheduled jobs that we did not envision at first. Such results, which are counter-intuitive, clearly illustrates the importance of validating the behaviours of scheduling algorithms through simulations before envisioning their real deployment.

Comparing the replication policies, we can see that *Replicate*3 and *Replicate*10 present similar results, and that the FullReplicate gains are almost double compared to the partial replication policies with a cost of doubling the size of the data transfers as well. We can deduce that replicating data-sets before applying the LocalityBased placement policy is beneficent from the users point of view, but deciding how much replication the system should do is not trivial. Going from 3 to 10 replicas does not seem to improve much the quality of service while doubling the cost in terms of data transfers, and duplicating the datasets everywhere almost halves the mean waiting time and bounded slowdown compared to the standard *Qarnot* scheduler, at a cost of multiplying by 5 the total size of data transfers.

Finding a good data-sets/jobs scheduling policy for *Qarnot Computing* is still an ongoing action. Our goal through this preliminary study was to illustrate the use of our framework on a concrete scenario, and to show how such a simulator would help to drive the design of an improved scheduling strategy.

6 Related simulation tools

We described in this paper a novel simulation tool for easily designing and testing scheduling strategies on edge computing platforms. We motivated the

huge effort of building a new simulator using adequate tools for modeling the processing and memory units and the network topology. We discuss briefly below the main competitors and argument for our simulator.

Some simulators have constraints that would prevent us to correctly simulate a platform such as the *Qarnot*'s one. For example, EmuFog [16] does not support hierarchical fog infrastructures, whereas *Qarnot* infrastructure is inherently hierarchical. Other simulators such as iFogSim [13], EdgeCloudSim [22] and IOTsim [25], are simulation frameworks that enable to simulate fog computing infrastructures and execute simulated applications on top of it. These are the closest work to ours and all rely on the CloudSim simulator. However, CloudSim, while widely used to validate algorithms and applications in different scientific publications, is based on a top-down viewpoint of cloud environments. That is, to the best of our knowledge, there are no articles that properly validate the different models it relies on. On the contrary, our simulator is built on top of SimGrid, which has been validated in many publications [2] and allows finer-grained simulations, as explained in Section 2.1.

7 Concluding remarks and future steps

We presented in this paper a dedicated toolkit to evaluate scheduling policies in edge computing infrastructures. Its integration into a simulator leads to a complete managing system for edge computing platforms that focuses on the evaluation of scheduling strategies.

While its complete validation is still an ongoing work, this toolkit already enables researchers/engineers to easily evaluate existing load balancing and placement strategies. It may also serve at developing and testing new strategies thanks to its modular and clear interface.

To assess the interest of such simulator, we instantiated the toolkit to simulate the whole edge platform of the *Qarnot Company* based on smart heaters. As a use case, we investigated several scheduling strategies and compared them to the actual policy implemented in the *Qarnot* platform. As expected, we showed that the best strategies are those which take into account locality and replication of data-sets.

In a future work, we envision designing an automatic and probabilistic injector of machine and network failures based on statistical studies of the platform logs and learning techniques.

Acknowledgments

This work was supported by the ANR Greco project and by AUSPIN with the International Exchange Program for undergraduate students from University of São Paulo.

References

[1] Qarnot computing. https://www.qarnot.com.

[2] Simgrid publications. http://simgrid.gforge.inria.fr/publications.html.

[3] A. Ahmed and E. Ahmed. A survey on mobile edge computing. In *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–8, Jan 2016.

[4] Farah Ait Salaht, Frédéric Desprez, Adrien Lebre, Charles Prud'Homme, and Mohamed Abderrahim. Service Placement in Fog Computing Using Constraint Programming. In *SCC 2019 - IEEE International Conference on Services Computing*, pages 1–9, Milan, Italy, July 2019. IEEE.

[5] Bill Allcock, Joe Bester, John Bresnahan, Ann L Chervenak, Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, and Steven Tuecke. Data management and transfer in high-performance computational grid environments. *Parallel Computing*, 28(5):749–771, 2002.

[6] David P Anderson. Boinc: A system for public-resource computing and storage. In *proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10. IEEE Computer Society, 2004.

[7] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[8] Flavio Bonomi, Rodolfo A. Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing, MCC@SIGCOMM 2012, Helsinki, Finland, August 17, 2012*, pages 13–16, 2012.

[9] A. Brogi and S. Forti. QoS-Aware Deployment of IoT Applications Through the Fog. *IEEE Internet of Things Journal*, 4(5):1185–1192, Oct 2017.

[10] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, June 2014.

[11] Bruno Donassolo, Ilhem Fajjari, Arnaud Legrand, and Panayotis Mertikopoulos. Fog Based Framework for IoT Service Provisioning. In *IEEE CCNC*, January 2019.

[12] Pierre-François Dutot, Michael Mercier, Millian Poquet, and Olivier Richard. 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.

[13] Harshit Gupta, Amir Vahid Dastjerdi, Soumya Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *Software: Practice and Experience*, 06 2016.

[14] Franz C. Heinrich, Tom Cornebize, Augustin Degomme, Arnaud Legrand, Alexandra Carpen-Amarie, Sascha Hunold, Anne-Cécile Orgerie, and Martin Quinson. Predicting the Energy Consumption of MPI Applications at Scale Using a Single Node. In *Cluster 2017*, Hawaii, United States, September 2017. IEEE.

[15] A. Lebre, J. Pastor, A. Simonet, and M. Südholt. Putting the next 500 vm placement algorithms to the acid test: The infrastructure provider viewpoint. *IEEE Transactions on Parallel and Distributed Systems*, 30(1):204–217, Jan 2019.

[16] Ruben Mayer, Leon Graser, Harshit Gupta, Enrique Saurez, and Umakishore Ramachandran. Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures. In *FWC*, pages 1–6. IEEE, 2017.

[17] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262, 2015.

[18] Mohammed Islam Naas, Philippe Raipin Parvedy, Jalil Boukhobza, and Laurent Lemarchand. iFogStor: An IoT Data Placement Strategy for Fog Infrastructure. In *ICFEC'17*, pages 97–104, 2017.

[19] Millian Poquet. *Simulation approach for resource management. (Approche par la simulation pour la gestion de ressources)*. PhD thesis, Grenoble Alpes University, France, 2017.

[20] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016.

[21] Olena Skarlat, Matteo Nardelli, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Optimized IoT Service Placement in the Fog. *SOC*, 11(4):427–443, Dec 2017.

[22] C. Sonmez, A. Ozgovde, and C. Ersoy. Edgecloudsim: An environment for performance evaluation of edge computing systems. In *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 39–44, May 2017.

[23] Ye Xia, Xavier Etchevers, Loïc Letondeur, Thierry Coupaye, and Frédéric Desprez. Combining Hardware Nodes and Software Components Ordering-based Heuristics for Optimizing the Placement of Distributed IoT Applications in the Fog. In *Proc. of the ACM SAC*, pages 751–760, 2018.

[24] Ashkan Yousefpour, Ashish Patil, Genya Ishigaki, Jason P. Jue, Inwoong Kim, Xi Wang, Hakki C. Cankaya, Qiong Zhang, and Weisheng Xie. QoS-aware Dynamic Fog Service Provisioning. 2017.

[25] Xuezhi Zeng, Saurabh Kumar Garg, Peter Strazdins, Prem Prakash Jayaraman, Dimitrios Georgakopoulos, and Rajiv Ranjan. Iotsim. *J. Syst. Archit.*, 72(C):93–107, January 2017.

[26] Ben Zhang, Nitesh Mor, John Kolb, Douglas Chan, Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee, and John Kubiatowicz. The Cloud is Not Enough: Saving IoT from the Cloud. In *HotStorage*, 2015.