Investigating Placement Challenges in Edge Infrastructures through a Common Simulator (extended version)

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 (extended version)

Adwait Bauskar[3], Anderson Da Silva[1,2], Adrien Lebre[3], Clément Mommessin[2], Pierre Neyron[2], Yanik Ngoko[4], Yoann Ricordel[4], Denis Trystram[2], and Alexandre Van Kempen[3]

[1]Institute of Mathematics and Statistics, University of São Paulo, Brazil
[2]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France
[3]Inria, LS2N, UMR 6004, IMT Atlantique, Nantes, France
[4]Qarnot Computing, Montrouge, France

February 20, 2020

Abstract

Scheduling computational jobs with data-sets dependencies is an important challenge of 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– perform fair comparisons between different proposals.

In this paper, we propose to address this limitation by presenting a simulation engine dedicated to the evaluation and comparison of scheduling and data movement policies for edge computing use-cases. Built upon the Batsim/SimGrid toolkit, our tool includes a plug-in system that allows researchers to add new models in order to cope with the diversity of edge computing devices. Moreover, it includes an injector that allows the simulator to replay a series of events captured in real infrastructures. We demonstrate the relevance of such a simulation toolkit by studying 2 scheduling strategies with 4 data movement policies on top of a simulated version of the Qarnot Computing platform, a production edge infrastructure based on smart heaters. We chose this use-case as it illustrates the heterogeneity as well as the uncertainties of edge infrastructures.

Our ultimate goal is to gather industry and academics around a common simulator so that efforts made by one group can be factorized by others.

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 [18] (NFV) have been accelerating the deployment of Cloud Computing-like capabilities at the edge of the Internet. Referred to as the Edge Computing [21] paradigm, the main objective is to perform on demand computations close to the place where the data are produced and analyzed to mitigate data exchanges and to avoid too high latency penalties [28]. 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, 8, 12, 19, 22, 25, 26], 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, available solutions have been evaluated either using *ad hoc* simulators or through 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 [16], we claim that a dedicated simulator toolkit to help researchers investigate Edge scheduling strategies should be released soon. Indeed, we claim that using placement simulators for Cloud Computing is not appropriate to study Edge challenges. Besides resource heterogeneity, network specifics (latency, throughput), and workloads, Edge Computing infrastructures differ from Cloud Computing platforms because of the uncertainties: connectivity between resources is intermittent, storage/computation resources are more heterogeneous and can join or leave the infrastructure at any time, for an unpredictable duration. In other words, a part of the infrastructure can be isolated or unavailable for minutes/hours preventing accessing some data-sets or assigning new computations.

In this article, we present several extensions we implemented on top of the Batsim/SimGrid toolkit [10, 13] to favor fair evaluations and comparisons between various scheduling strategies for Edge infrastructures. In particular, we developed an external module to allow injecting in the simulation any type of unforeseen events that could occur (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 other available Edge simulators [23, 27] 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, favors straightforward comparisons of different strategies, even if they have been implemented by different research groups;

- Batsim promotes separation of concerns and enables the decoupling between the core simulator and the scheduler. Moreover, Batsim provides APIs in different languages (including Python and C++) that makes the development of a scheduling strategy accessible for a large number of researchers;
- The accuracy of the internal models (computation and network) of SimGrid has been already validated [11, 24] and extensively used [2].
- SimGrid provides a plug-in mechanism, which is of particular interest to deal with the diversity of Edge devices: it lets researchers add new models of specific Edge facilities without requiring intrusive modifications into the simulation engine.

By extending Batsim to the Edge paradigm, we target a tool that will enable researchers/engineers to re-evaluate major state-of-the-art load balancing strategies. 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], which have several characteristics in common with Edge platforms can be slightly revised to cope with Edge specifics.

To demonstrate the relevance of our proposal, we discuss several simulations we performed for the *Qarnot Computing* [1] use-case. The infrastructure of *Qarnot Computing* is composed of 3,000 diskless machines (smart heaters) distributed across several sites in France. Each computing resource can be used remotely as traditional Cloud Computing capabilities or locally to satisfy data processing requirements of IoT devices 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 data-sets dependencies.

After giving a description of our Edge platform simulator, we present how the *Qarnot* infrastructure has been instantiated on top of it, how we used our injector to simulate the *Qarnot* workload and, finally, how we evaluated different strategies for job placement and data movements.

The strategies presented in the article are simple. They aim to illustrate what can be done without important efforts. More advanced strategies can be analyzed in the same manner. We are, for instance, investigating more advanced strategies that consider pulling data-sets from other Edge resources rather than from the centralized storage system of the *Qarnot Computing* infrastructure.

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 [13] delivers a high-level API to facilitate the development of batch scheduling algorithms to be simulated on top of SimGrid [10]. Thus, our proposal relies on tools that have already been validated by our community.

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 state-of-the-art simulation toolkit that enables the simulation of distributed systems. SimGrid's relevance in terms of performance and validity has been backed-up by many publications [2]. In addition to providing the program to be evaluated, performing simulations with SimGrid requires writing a platform specification and interfacing the program to simulate. SimGrid enables the description of complex platforms, such as hierarchical infrastructures composed of many interconnected devices with possibly highly heterogeneous profiles, such as the edge ones.

2.1.2 Batsim and the decision process

Batsim [13] is a simulator engine built on top of SimGrid. It proposes a specialized API to help researchers design and analyze jobs and I/O scheduling systems. Such systems are for instance Batch Schedulers *a.k.a.,* Resource and Jobs Management Systems, in charge of managing resources in large-scale computing centres. Batsim allows researchers to simulate the behaviour of a computational platform in which workloads are 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, 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 Batsim makes straightforward. Internally, the decision process (i) reacts to the simulation events received from Batsim, (ii) takes decisions according to the given scheduling algorithm, and (iii) 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.

For more details on Batsim and SimGrid mechanisms, we invite the reader to refer to Chapter 4 of Poquet's manuscript [20].

2.2 Extensions

To ease the study of scheduling strategies for Edge platforms, we have been working on a couple of extensions for Batsim. We present in this section those which are already available, namely the events injector and the storage controller. Modifications made in Batsim[1] and its Python API[2] for this work are available in a dedicated branch of their main repository. Besides, we presend the plug-in mechanism of SimGrid that researchers can leverage to provide models of particular Edge devices.

2.2.1 External events injector

To simulate the execution of an 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 impact the behaviour of the platform during the execution 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 becomes unexpectedly unavailable for a period of time, due to a failure or action (e.g., from a local user) occurring at the edge.

The mechanism we implemented replays external events that occurred at a given time, and lets the main process of Batsim handle them. Batsim updates the state of the platform, and then forwards the events to the decision process. An event is represented as a JSON object composed of two mandatory fields: a *timestamp* that indicates when the event should occur, and the *type* of the event. Depending on the type of event, other fields can complement the event description, such as for instance the name of the unavailable resource, the new value of an environment parameter such as the network bandwidth, or anything of interest to the decision process. External events are injected in Batsim by one of its internal processes, which reads from an input file the aforementioned JSON objects (one per line).

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

2.2.2 Storage Controller

The Storage Controller is a Python module that manages the storage entities and supervises data transfers during the simulation.

At the beginning, the Storage Controller reads an input file that contains the description of each data-set (i.e., one JSON object per line describing whether

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

a particular data set is present or not on a given storage entity). During the simulation, the Storage Controller keeps track of the on-going data transfers, the available data-sets in each storage entity and manages all aspects related to caching policies of these entities. A simple API is exposed to the decision process to ask, for example, a data-set to be copied in another storage or to retrieve the list of storage having a copy of a given data-set.

2.2.3 SimGrid plug-ins

When designing an Edge platform simulator, it is a nonsense to foresee all the models and devices that may compose a platform, as they are highly heterogeneous with, for example, cameras, sensors, actuators, etc. In this context, having a plug-in system such as the one proposed by SimGrid helps researchers/engineers integrate new models according to the devices they want to consider.

On the other hand, it is not practically feasible to implement in Batsim a generic way of exposing to the scheduler any function designed in such a plug-in. For this reason, modifications in the code of Batsim are required to extend the communication protocol and exchange information between a plug-in and the decision process, as explained in Section 4 for the case of the *Qarnot* platform.

3 Case study: the Qarnot Computing platform

We present in this section the *Qarnot Computing* infrastructure we use to demonstrate the relevance of our simulation tool.

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 sites in France. As of writing this paper, the whole platform is composed of about 1,000 computing devices hosting about 3,000 diskless machines, and is growing quickly. On each of the 20 geographical sites, there is a NFS-based storage that enables diskless machines to manipulate data. 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 NFS server. The latency between a computing machine and the NFS server 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 symmetric bandwidth to the Internet, and about 10 ms latency to French data centers used by *Qarnot* to host control and monitoring services, the central storage system, and gateways to its distributed infrastructure.

On a daily basis, *Qarnot* computing solution processes from a few hundred to several thousands of batch jobs and thousands of cores are provisioned for

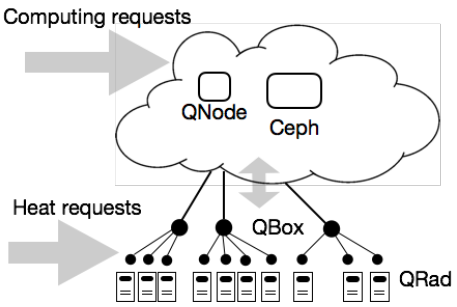

Figure 1: Scheme of the Qarnot platform.

dedicated corporate customers. From the data movement viewpoint, up to tens of GB of data are replicated from central storage to Edge Computing sites.

3.2 Platform Organization and Terminology

The jobs and resources manager of the *Qarnot* platform is based on a hierarchy of 3 levels, as shown in Figure 1: the *Qnode*-, the *QBox*- and the *QRad*-level. The QNode is a root node, a "global" server that takes placement decisions for the whole platform. It can be viewed as a load balancer for the platform. On the second level are the QBoxes, "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 diskless 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 the CEPH are in the Cloud while QBoxes are distributed over smart buildings of several cities, while QRads among a building are distributed in different rooms.

The *Qarnot* platform receives two types of user requests: requests for computing 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 and a Docker image needed to execute their jobs to the CEPH. Then, they submit their job, denoted by *QTask*, to the QNode. A QTask can be decomposed as a bag of several *instances* that share the same Docker image and data dependencies, but with different command arguments. 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 an Edge infrastructure: the computation capacity does not simply appear or disappear but also vary according to the heating

needs.

3.3 Principle of the Actual Scheduling Policy

QTasks submitted to the platform are scheduled onto QMobos through two 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, "dummy" compute-intensive programs are executed to keep the QRad warm.

Modelling such an infrastructure in order to identify improvement opportunities and analyze new scheduling strategies is something possible with our Batsim extensions.

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.

Each QMobo of the platform is simulated as a SimGrid host as they are the only computing units of the platform. We keep the same hierarchical structure of the *Qarnot* platform: QMobos belonging to the same QRad are aggregated in the same SimGrid zone (representing a network). Similarly, all QRads of a same QBox are aggregated in the same zone, as well as 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. Similarly, 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 instances belonging to the same QTask.

As temperature plays an important role in the platform and the scheduling decisions, we leveraged the plug-ins mechanism of SimGrid to implement our own model. Built on top of the already-existing energy plug-in [15], our plug-in computes the temperature of a QRad and its ambient air from the energy consumption of the computing resources of the QRad and other physical parameters. Regarding the simulation of heating requests, each change of the target temperature of a QRad is simulated as an external event injected in the simulation, as well as the outside temperature of the cities where the QRads

are deployed, injected every hour. Thus, we modified Batsim to relay these external events to the plug-in, and we modified the communication protocol of Batsim to periodically forward the ambient air temperature of each QRad to the scheduler.

Finally, the schedulers of the QNode- and QBox-level were implemented using Batsim's Python API and both live in the same process, along with the Storage Controller.

A log extractor was built to generate all the input files from real logs of the *Qarnot* platform, for a given time period. These are all the files given as input to Batsim and the decision process for the simulation: an XML file describing the platform and several JSON files containing the list of jobs to execute, the list of data-sets in the CEPH and the list of external events. Since we want to simulate an exact time period, we added a special external event that enforces the simulation to stop at a particular time.

5 Experiments

In this section we discuss the simulations we performed to investigate the *Qarnot Computing* use case. We underline that, due to privacy reasons, we cannot provide access to the log extractor and the *Qarnot* logs used for the experiments. However, the code of the evaluated schedulers is available in a dedicated branch of Batsim's Python API repository[3].

We conducted two kind of experiments. The first aimed to compare the standard scheduling policy used in the real *Qarnot* platform with a policy based on locality of the data-sets. The second experiment enabled us to study the impact of replication policies for the data-sets that are uploaded on the platform (i.e., how they affect the scheduling decisions).

For both experiments, we compared the quality of the produced schedules using the total number of transfers that occurred and the total data transferred in GB, as well as the waiting time of the instances. For an instance, the waiting time denotes the difference between its starting time and submission time.

We extracted 4 different simulation inputs corresponding to logs of the *Qarnot* platform for a 1-week period each. Since the simulation and the scheduling algorithms are deterministic, we ran one simulation with each combination of scheduler and workload. Each simulation took less than 40 minutes to run, with about 15% of the time spent in the decision process.

The considered workloads contained between 5,000 and 9,000 instances and between 40 and 60 different data-sets. In each workload there was at least one data-set used by 50% of the instances, and for the 3rd workload up to 9 data-sets were used by 700 instances. Thus, a scheduler taking into account data-locality and a good replication policy could greatly improve the quality of the schedules compared to standard scheduling decisions.

Regarding the processing time of the instances, we notice that there is at least 75% of instances of each workload having a short processing time (smaller

[3] `https://gitlab.inria.fr/batsim/pybatsim/tree/temperature`

than 10 minutes); and a maximum processing time of almost 10 hours for the first workload.

5.1 Scheduling and Replication Policies

Along with the real *Standard Qarnot* scheduler that serves as a baseline for our experiments (see Section 3.3), we also implemented a variant using the data-locality to take scheduling decisions at the QNode-level, denoted by *LocalityBased*. Upon dispatching instances, LocalityBased gives priority to the QBoxes already having the data-set dependencies of the QTask on their storage disk. This variant aims at taking benefit from the data locality and reducing the data transfers.

To evaluate the impact of data placement on the scheduling decisions, we also implemented three variants of replication policies upon the submission of QTasks. The first two variants, denoted by *Replicate3* and *Replicate10*, respectively replicates the data-dependencies of a submitted QTask on the 3 and 10 least loaded QBox disks among the 20 QBoxes in the platform, before applying the LocalityBased scheduling algorithm. These two variants aim at reducing the waiting time of the instances by providing more QBox candidates for the LocalityBased dispatcher.

The last variant, denoted by *FullReplicate*, instantaneously replicates all data-set dependencies on all QBox disks upon the submission of a QTask. Even if it is unrealistic, this variant aims at visualizing the behaviours of the standard scheduling policy without having any impact caused by the data placements.

5.2 Exploiting Data Locality

As Figure 2 shows, the main goal of the LocalityBased scheduler is fulfilled: dispatching instances on QBoxes already having the data-set dependencies on their disk permits to reduce the number of transfers by about 40%, and between 30 and 65% the total data transferred, compared to Standard.

Figures 3 to 6 shows the waiting time distribution achieved by each scheduler respectively for each workload, separated in 3 intervals for better clarity. As one can see, for each scheduler more than 50% of the instances waited less than one second before starting their execution. Comparing the behaviour of Standard and LocalityBased, we do not observe a big difference in the distribution of the waiting times. This is confirmed by the average value over all instances of 54.7 seconds for Standard and 54.4 for LocalityBased.

We also notice that a few instances waited a long time (around 1,455 or 2,910 seconds) before starting their execution. This is due to the long transfer time of one of their data dependencies that was as large as 36 GB, while every other data-set is smaller than 5 GB.

Thus, we conclude that taking into account only the data locality is not sufficient to have good waiting time performances, and that it is crucial to also consider the time spent in data transfer before taking scheduling decisions. In this regard, we are currently working to extend the Storage Controller to

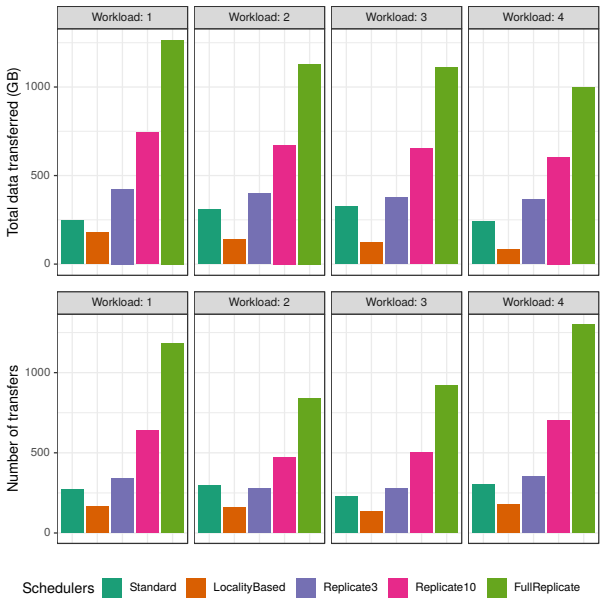

Figure 2: Number of transfers and total data transferred in GB, for all schedulers and workloads.

estimate the transfer time of a data-set to a given storage entity, which will be used as additional information by the schedulers.

5.3 Varying the Data Placement Policies

When comparing the two schedulers with replication policies (Replicate3 and Replicate10) with LocalityBased, which does not perform replication upon instance submissions, Figures 3 to 6 shows that the behaviour is similar to Standard and LocalityBased with more than 50% of the instances having a waiting time smaller than 1 second, and a few instances impacted with the large data-set with a waiting time of 1455 seconds.

Comparing the average waiting time shows that more replications permits to reduce the mean waiting times of the instances, at a cost of more data transfers, as depicted in Figure 2. More precisely, the mean waiting time of the instances decreases from 54.4 to 47.3 seconds for Replicate3 and 40.3 for Replicate10. However, this improvement in the mean waiting time does not seem sufficient compared to the increase in data transfers (from 120 GB to 376 GB for Replicate3 and 652 GB for Replicate10) and the engineering work that would be required if such replication solution were to be implemented on the real platform.

Regarding the FullReplicate scheduler, where data transfers are instantaneous, we observe that it greatly outperforms all other schedulers in terms of

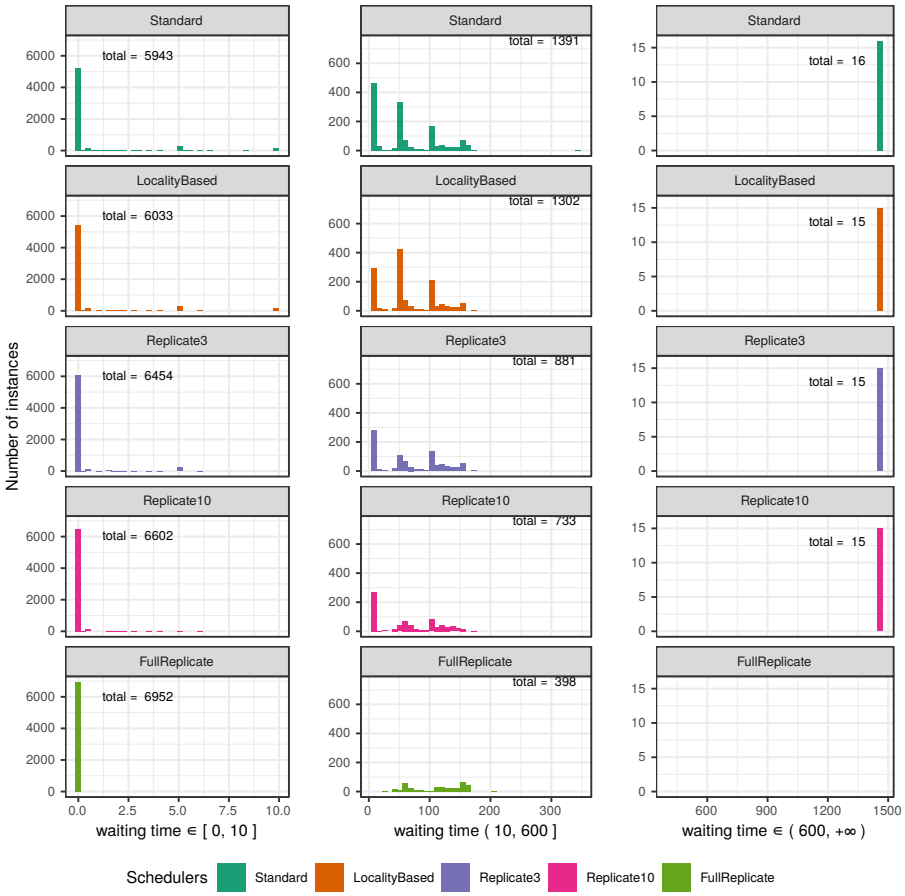

Figure 3: Waiting time distribution (in seconds) of all instances of workload 1, for all schedulers.

waiting time, with a mean at 30 seconds, at a cost of more than 1,000 GB of data transferred. These results confirms the observations made previously that the transfer time of data-sets represents most of the waiting time of the instances, and that work should be done to reduce it as much as possible. In this regard, we are also working with engineers of *Qarnot Computing* to enable retrieval of data-sets between QBox disks, which is expected to improve transfer times compared to the current solution where only the centralized storage server can supply the data-sets.

Finally, we recall that our goal through this study was not to find the best scheduling algorithm but to illustrate the use of our simulation toolkit on a concrete scenario, and to demonstrate how such a simulator would help to drive the design of scheduling and data placement strategies.

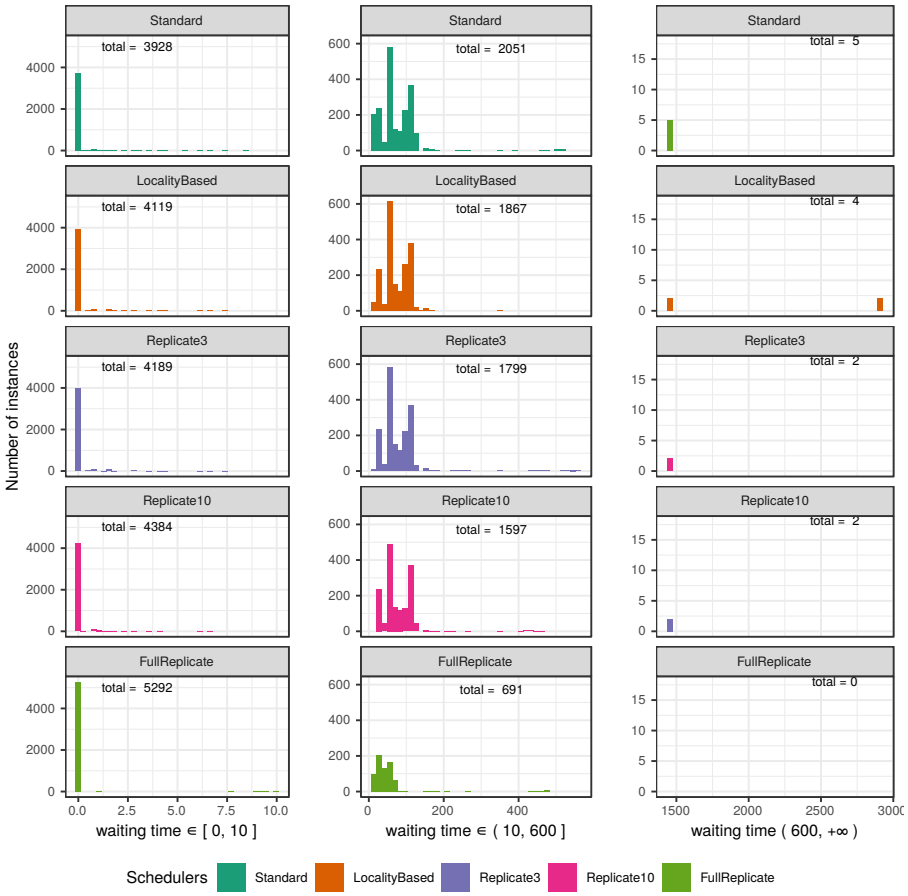

Figure 4: Waiting time distribution (in seconds) of all instances of workload 2, for all schedulers.

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 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 [17] does not support hierarchical fog infrastructures, whereas *Qarnot* infrastructure is inherently hierarchical. Other simulators such as iFogSim [14], EdgeCloudSim [23] and IOTSim [27], are simulation frameworks that enable to simulate Fog or Edge Computing infrastructures and execute simulated applications on top of

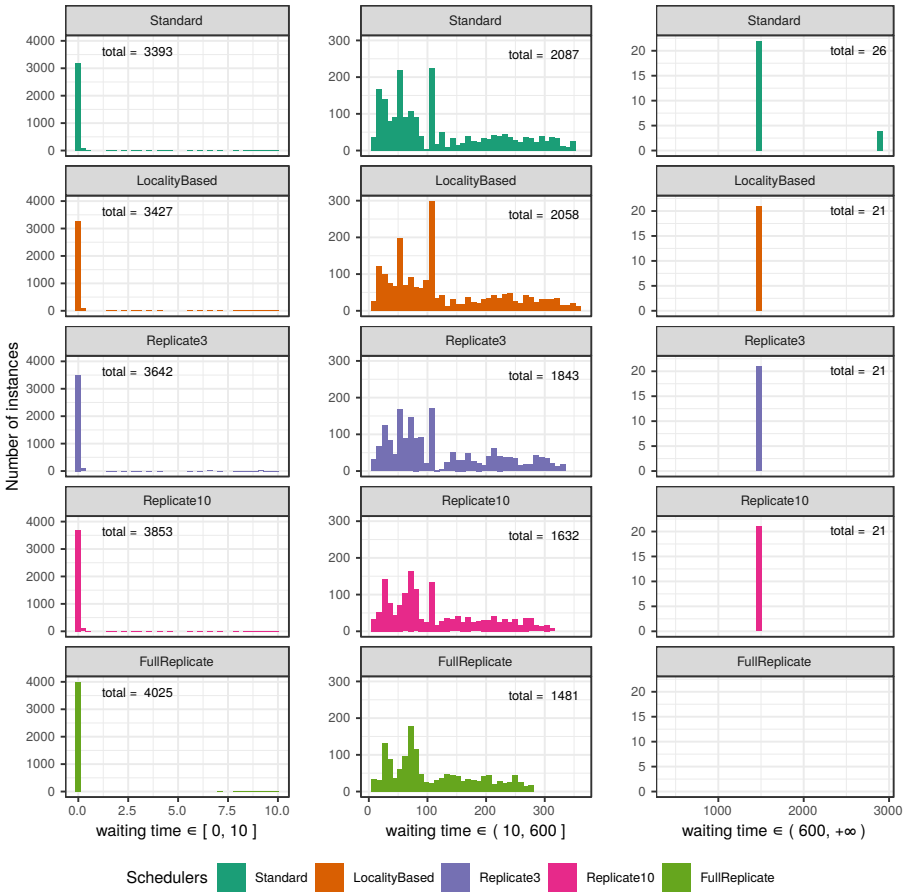

Figure 5: Waiting time distribution (in seconds) of all instances of workload 3, for all schedulers.

it. These are the closest work to ours and all rely on the CloudSim simulator [9]. However, while widely used to validate algorithms and applications in different scientific publications, CloudSim is based on a top-down viewpoint of cloud environments. To the best of our knowledge, there are no articles that properly validate the different models it relies on, as opposed to SimGrid, which has been validated in many publications [24, 11, 2] and allows finer-grained simulations. Moreover, implementing different placement and scheduling policies in CloudSim and related simulators requires to directly modify the code of the simulator, while it is part of a separate decision process that communicates with Batsim in our solution, making it more versatile and user-friendly.

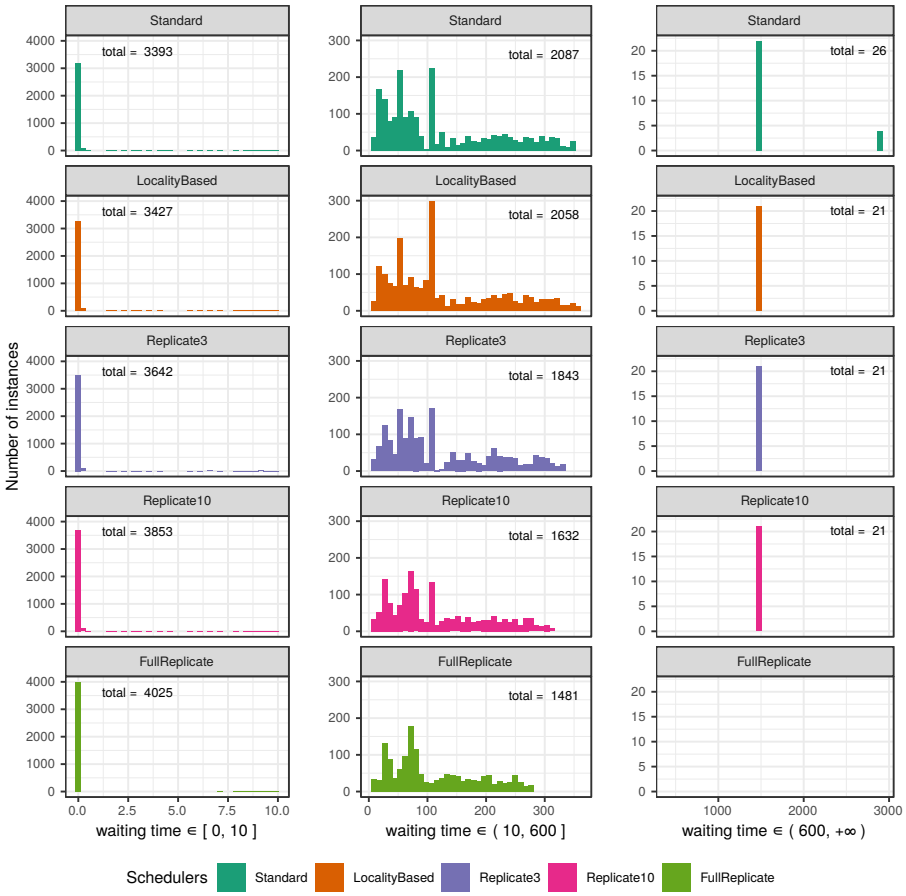

Figure 6: Waiting time distribution (in seconds) of all instances of workload 4, for all schedulers.

7 Concluding remarks and future steps

We presented in this paper a dedicated toolkit to evaluate scheduling and data placement policies in Edge Computing infrastructures. Its integration into a simulator leads to a complete management system for Edge Computing platforms that focuses on the evaluation of scheduling strategies.

While more extensions are still under development, 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 Computing* 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 strategy is the one that takes into account locality. We also showed replication of data-sets is efficient, but it should be mitigate by heavy data transfer times.

We envision now to design an automatic and probabilistic injector of machine and network failures based on statistical studies of the platform logs and learning techniques. We are also currently working to extend the Storage Controller to estimate the transfer time of a data-set between two storage entities and to implement scheduling strategies benefiting from this feature.

Acknowledgments

This work was supported by the ANR Greco project 16-CE25-0016-01 and by AUSPIN with the International Student Exchange Program from the University of São Paulo.

References

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

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

[3] Ahmed, A., Ahmed, E.: A survey on mobile edge computing. In: 2016 10th International Conference on Intelligent Systems and Control (ISCO). pp. 1–8 (Jan 2016). https://doi.org/10.1109/ISCO.2016.7727082

[4] Ait Salaht, F., Desprez, F., Lebre, A., Prud'Homme, C., Abderrahim, M.: Service Placement in Fog Computing Using Constraint Programming. In: SCC 2019 - IEEE International Conference on Services Computing. pp. 1–9. IEEE, Milan, Italy (Jul 2019), `https://hal.archives-ouvertes.fr/hal-02108806`

[5] Allcock, B., Bester, J., Bresnahan, J., Chervenak, A.L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D., Tuecke, S.: Data management and transfer in high-performance computational grid environments. Parallel Computing **28**(5), 749–771 (2002)

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

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

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

[9] Calheiros, R., Ranjan, R., Beloglazov, A., De Rose, C., Buyya, R.: Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software Practice and Experience **41**, 23–50 (01 2011). https://doi.org/10.1002/spe.995

[10] Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. Journal of Parallel and Distributed Computing **74**(10), 2899–2917 (Jun 2014). https://doi.org/10.1016/j.jpdc.2014.06.008

[11] Degomme, A., Legrand, A., Markomanolis, G., Quinson, M., Stillwell, M.L., Suter, F.: Simulating MPI applications: the SMPI approach. IEEE Transactions on Parallel and Distributed Systems **28**(8), 14 (Aug 2017). https://doi.org/10.1109/TPDS.2017.2669305, `https://hal.inria.fr/hal-01415484`

[12] Donassolo, B., Fajjari, I., Legrand, A., Mertikopoulos, P.: Fog Based Framework for IoT Service Provisioning. In: IEEE CCNC (Jan 2019)

[13] Dutot, P.F., Mercier, M., Poquet, M., Richard, O.: 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)

[14] Gupta, H., Vahid Dastjerdi, A., Ghosh, S., Buyya, R.: 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). https://doi.org/10.1002/spe.2509

[15] Heinrich, F.C., Cornebize, T., Degomme, A., Legrand, A., Carpen-Amarie, A., Hunold, S., Orgerie, A.C., Quinson, M.: Predicting the Energy Consumption of MPI Applications at Scale Using a Single Node. In: Cluster 2017. IEEE, Hawaii, United States (Sep 2017), `https://hal.inria.fr/hal-01523608`

[16] Lebre, A., Pastor, J., Simonet, A., Südholt, M.: 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). https://doi.org/10.1109/TPDS.2018.2855158

[17] Mayer, R., Graser, L., Gupta, H., Saurez, E., Ramachandran, U.: Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures. In: FWC. pp. 1–6. IEEE (2017)

[18] Mijumbi, R., Serrat, J., Gorricho, J.L., Bouten, N., De Turck, F., Boutaba, R.: Network function virtualization: State-of-the-art and research challenges. IEEE Communications Surveys & Tutorials **18**(1), 236–262 (2015)

[19] Naas, M.I., Parvedy, P.R., Boukhobza, J., Lemarchand, L.: iFogStor: An IoT Data Placement Strategy for Fog Infrastructure. In: ICFEC'17. pp. 97–104 (2017)

[20] Poquet, M.: Simulation approach for resource management. (Approche par la simulation pour la gestion de ressources). Ph.D. thesis, Grenoble Alpes University, France (2017), https://tel.archives-ouvertes.fr/tel-01757245

[21] Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: Vision and challenges. IEEE Internet of Things Journal **3**(5), 637–646 (Oct 2016). https://doi.org/10.1109/JIOT.2016.2579198

[22] Skarlat, O., Nardelli, M., Schulte, S., Borkowski, M., Leitner, P.: Optimized IoT Service Placement in the Fog. SOC **11**(4), 427–443 (Dec 2017)

[23] Sonmez, C., Ozgovde, A., Ersoy, C.: Edgecloudsim: An environment for performance evaluation of edge computing systems. In: 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC). pp. 39–44 (May 2017). https://doi.org/10.1109/FMEC.2017.7946405

[24] Velho, P., Schnorr, L., Casanova, H., Legrand, A.: On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations. ACM Transactions on Modeling and Computer Simulation **23**(4) (Oct 2013), https://hal.inria.fr/hal-00872476

[25] Xia, Y., Etchevers, X., Letondeur, L., Coupaye, T., Desprez, F.: 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. pp. 751–760 (2018)

[26] Yousefpour, A., Patil, A., Ishigaki, G., Jue, J.P., Kim, I., Wang, X., Cankaya, H.C., Zhang, Q., Xie, W.: QoS-aware Dynamic Fog Service Provisioning (2017)

[27] Zeng, X., Garg, S.K., Strazdins, P., Jayaraman, P.P., Georgakopoulos, D., Ranjan, R.: Iotsim. J. Syst. Archit. **72**(C), 93–107 (Jan 2017). https://doi.org/10.1016/j.sysarc.2016.06.008, https://doi.org/10.1016/j.sysarc.2016.06.008

[28] Zhang, B., Mor, N., Kolb, J., Chan, D., Lutz, K., Allman, E., Wawrzynek, J., Lee, E., Kubiatowicz, J.: The Cloud is Not Enough: Saving IoT from the Cloud. In: HotStorage (2015)