Master of Science in Informatics at Grenoble
Master Informatique
Specialization Parallel, Distributed and Embedded Systems

Investigating Job Allocation Policies in Edge Computing Platforms

Anderson Andrei DA SILVA

June 24th, 2019

Research project performed at Laboratoire d'Informatique de Grenoble

Under the supervision of:
Prof. Denis TRYSTAM

Defended before a jury composed of:
Prof. Martin HEUSSE
Prof. Christophe CÉRIN
Prof. Hubert GARAVEL

June 2019

Abstract

Fog/Edge computing infrastructures have as one of the most important challenges the allocation of computational jobs with data-sets dependencies in an efficiently way. In this thesis, we present the usage of a simulated Edge platform and several metrics exemplifying how to develop and compare different scheduling strategies. More precisely, we utilize 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 features if needed.

The implementation is 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. We are able to develop and simulate the current Qarnot's job allocation strategies, to propose and evaluate different approaches for different metrics.We also propose an analysis of their workloads and data sets dependencies.

Finally, this thesis discusses the current state of the art of Edge and Cloud computing and summarizes some future directions that have been raised in the literature.

Résumé

Les infrastructures de calcul *fog* et *edge* ont comme important défi l'allocation de tâches de calcul, avec les jeux de données dont elles dépendent, de manière efficace. Dans cette thèse, nous présentons l'utilisation d'une plateforme Edge simulée pour démontrer comment développer et comparer différentes stratégies d'ordonnancement. Plus précisément, nous utilisons un projet en cours regroupant des partenaires universitaires et une entreprise de haute technologie qui a pour but de délivrer un outil d'évaluation des politiques d'ordonnancement dans une plateforme de calcul *Edge*. Cet outil permet à la communauté de simuler diverses politiques et de personnaliser facilement des cas d'utilisations de checheurs/ingénieurs, via l'ajout de nouvelles fonctionnalités si besoin.

L'implémentation de cet outil est basé sur le simulateur Batsim/SimGrid, qui a été conçu pour évaluer des stratégies d'ordonnancement *batch* dans diverses infrastructures distribuées. Bien que la validation complète de notre outil de simulation est un travail en cours, nous démontrons sa pertinence par l'étude de différentes stratégies d'ordonnancement dans une version simulée de la plateforme de Qarnot Computing, une infrastructure de calcul *Edge* basée sur des radiateurs intelligents. Nous avons pu développer et simuler la politique d'allocation de tâches de Qarnot, pour proposer et évaluer différentes approches d'ordonnancement pour différentes métriques. Nous proposons également une analyse des ensembles de tâches exécutées sur la plateforme et de leurs dépendances de données.

Finalement, cette thèse discute l'état de l'art du calcul dans le *Cloud* et l'*Edge* et résume certaines orientations futures qui ont été soulevées dans le cadre du processus de la littérature.

Acknowledgement

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.

I would like to thank my advisors, Professor Denis Trystam for the opportunity, his supporting and teachings, and Professor Alfredo Goldman for indicated me for this Master Program. Also Dr. Yanik Ngoko for having proposed the subject of this thesis.

I would like to thank my family and friends for all the support, my fiancée Andréia, my mom Rosimeri, my father Alex and my brother Leonardo.

Finally, I would like to thank anyone in my work group, colleagues and also ones responsible for the administrative processes from USP and UGA.

Contents

— 1 —
Introduction

1.1 Edge Computing and the Internet of Things

The proliferation of Internet of Things (IoT) applications [8], as well as the advent of new technologies such as Mobile Edge computing [3], and Network Function Virtualization [27] (NFV) have been accelerating the deployment of Cloud Computing like capabilities at the edge of the Internet. Cloud computing has been targeted of centralized computation, where in general, the data is sent to the Cloud, computed there and the delivered to one that requested it. However, nowadays, mobile devices have considerable computation power embedded, hence, the usage of these for one hand has requested the delivery of processed information from the Cloud as soon as possible, and for the other hand, it has computed its own computation and then get available for more data processing much faster than previously. So, a paradigm has been growth in the aspect of data processing, storage and transfer, which is the one to utilize these devices between the users and the Cloud to perform all these computations [9]. Referred to as the Fog [10] or the Edge computing [37] 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 [45].

Among the open questions, our community should address to favor the adoption of such infrastructures is the computation/data placement problem *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 [43, 11, 40, 42, 28, 14, 5], 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. In addition to the resource heterogeneity, network specifics as (latency, throughput, etc.) 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. Hence, is necessary a tool to provide a structured Edge platform to allow studies.

1.2 An Edge Simulated Platform

Similarly to what has been proposed for the Cloud Computing paradigm [23], a dedicated simulator toolkit to help researchers investigate HPC scheduling strategies have been developed, Batsim [15]. Utilizing its decision maker component, as our goal, many scheduling policies have been developed to then be compared in terms of performance when applied in a use case. In particular, Batsim also provides an external module to inject any type of event that could occur during the simulation (e.g., a machine became unavailable at time t) and a Storage Controller, to supervise all transfers of data sets within the simulated platform. Also, there is a scientific instrument to study the behavior of large scale distributed systems such as Grids, Clouds, HPC or P2P systems, Simgrid [12]. It can be used to evaluate heuristics, prototype applications or even assess legacy MPI applications.

In this thesis we will present the Batsim/Simgrid toolkit that provide several extensions that make possible the construction of an Edge simulated platform and also its performance analyses. Hence, as we applied different policies into a use case, researchers can use it to study whether scheduling algorithms that have been proposed two decades ago in desktop computing platforms, volunteer computing and computational grids [7, 6, 16] reviewing to cope with edge specifics. While edge workloads differ from best effort jobs [1], desktop/volunteer computing and computational grids have several characteristics that are common to Fog/ Edge platforms.

1.3 Case Study

Although the validation of these extensions, as well as the integration of representative edge workloads is still on going, the first building blocks implemented enabled the study of 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 data sets dependencies. As far the simulation of the Qarnot platform was possible utilizing Batsim/SimGrid, it also provided us the possibility to apply different scheduling policies to the Qarnot's jobs.

1.4 Main Contributions

This thesis contributed with the schedulers that were developed. For that we connected all the current extensions that have been developed, including, the Storage Controller and the logs extractor. To compare different scheduling policies from different points of view, this thesis contributed with the design of experiments that allow the easy modification, execution and visualization of all results provided by the platform. These analyses was conducted by the study of the job's processing time distribution, the job's dependencies of data sets, the comparison

[1] Jobs managed under a best effort network, meaning that they obtain unspecified variable bit rate and latency and packet loss, depending on the current traffic load

among the workloads extracted and among the developed policies by several metrics. In addition, beyond the metrics provided by Batsim/Simgrid as waiting time, scheduling time and slowdown this work contributed with the addition of a classical metrics utilized in the literature: bounded slowdown.

Furthermore, this thesis contributed to the simulation of the whole Qarnot platform. In order to present details of its implementation, we will depict the whole platform giving an overview of the Edge placement simulator. After presenting the simulated platform built with Batsim/Simgrid, we will describe how the *Qarnot* infrastructure has been instantiated on top of it, how the injector was used to simulate the *Qarnot* workload and, finally, how was developed and evaluated different scheduling strategies for job placement and data movements.

With that results and all extensions connected as different components, a paper was submitted to the IEEE MASCOTS 2019 conference presenting this Edge Platform Simulator [13]. In addition, one can find the whole experimental structure of this project in the *uga-master-thesis* repository on GitHub [2], there are more figures and all scripts utilized for experiments.

1.5 Outline

The rest of the thesis is structured as follows. Chapter 2 presents related works. Chapter 3 gives an overview of the Bastim/SimGrid toolkit and the extensions utilized. Chapter 4 presents the *Qarnot Computing* use case. Chapter 5 describes how we simulated the use case. Chapter 6 presents concepts about job allocation and metrics to evaluate performance in the context of this thesis. Also depicts the algorithms developed based on the use case and its differences. Chapter 7 discusses analyses of the different scheduling strategies for the *Qarnot* platform. We also show investigations regarding jobs processing time and the data sets dependencies based on the simulation and the extracted logs. Concluding and future remarks are presented in Chapter 8.

[2] https://github.com/andersonandrei/uga-master-thesis

— 2 —
State of the Art

This thesis is related to several terms and concepts into the development of an edge simulated platform focused in the scheduling of HPC jobs. Hence we have been studied the state of the art of these concepts follow as cluster, grid and cloud computing, the growth of IoT and Mobile devices as a reasonable usage of these computing platforms. Therefore, Edge computing has been studied. Regarding the allocation of HPC jobs, we will present some studies about this kind of job and its allocation process from different possible goals and views. Running these jobs in platforms as cluster, grid, edge or cloud, is visible the necessity to use metrics to evaluate the performance for that processes and techniques. So, we will present some of these techniques as load balance of HPC jobs and the usage of virtualization and containers to manage this kind of platform. In addition, as the energy consumption has been a very important problem related with all of these concepts, we will present some related works that handle it. In the sequence we will present some recent statistics by scientific studies and by the viewpoint of companies that emphasizes the emergency and the requirement of edge computing. Finally, we will show some related simulated platforms providing comparisons with the one utilized and developed during this work.

2.1 Computing Platforms, Difference and Evolution

In a survey, Huang et.al [21] define three phases as the evolution of the computing platforms following the paradigm of data processing over the years. The mode of calculation has been changed along the time, and Cloud computing defines the current state. It shows this evolution process from the steps:

1. The original mode which for processing, gathered all the tasks to large-scale processors.
2. The distributed tasks processing mode based on the Internet.
3. The cloud computing mode for immediate processing.

Hameed Hussain et.al [22], defined as HPC categories, Cluster, Grid and Cloud computing platforms are conceptually similar, but its their main different features are emphasized as follow:

- on Clusters, the goal is to design an efficient computing platform that uses a group of commodity computer resources integrated through hardware, networks, and software to improve the performance and availability of a single computer resource, which

  - A modern one is made up of a set of commodity computers that are usually restricted to a single switch or group of interconnected switches within a single virtual local-area network (VLAN). In addition to executing compute-intensive applications, cluster systems are also used for replicated storage and backup servers that provide essential fault tolerance and reliability for critical parallel applications.
  - Allows extensions by incorporating load balancing, parallel processing, multi-level system management, and scalability methodologies.

- on Grid, the computing concept is based on using the Internet as a medium for the wide spread availability of powerful computing resources as low-cost commodity components . Computational grid can be thought of as a distributed system of logically coupled local clusters with non-interactive workloads that involve a large number of files. Emphasizing that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed, makes grid different from conventional HPC systems, such as cluster.

- on Cloud, is found a recent model for Information Technology (IT) services based on the Internet, and typically involves provision of dynamically scalable and often virtualized resources over-the-Internet. Typical cloud computing providers deliver common business applications online that are accessed through web service, and the data and software are stored on the servers. In addition, the cloud computing systems are difficult to model with resource contention (competing access to shared resources). Many factors, such as the number of machines, types of application, and overall workload characteristics, can vary widely and affect the performance of the system.

Focusing on Grid Computing, Qureshi et.al [33], presents it as a platform for virtual organizations and computing environments that was introduced in 1990s, which:

- Provides low-cost intelligent methodologies for sharing data and resources such as computers, software applications, sensors, storage space, and network bandwidth due to the necessity of reliable, pervasive, and high computing power.

- Depending on factors as operating system, amount of memory, CPU speed, number of resources, architecture types and so on, Grids can be generally classified as homogeneous or heterogeneous.

Several surveys present the definition of Cloud computing as composed of three kind of services [31, 22, 33]:

- Cloud Software as a Service (SaaS), which cloud providers offer software running on a cloud infrastructure.

- Cloud Platform as a Service (PaaS), which the cloud platform offers an environment for development and deployment of applications.

- Cloud Infrastructure as a Service (IaaS), which cloud providers manage computing resources such as storing and processing capability.

In addition, different deployment models have been adopted based on their variation in physical location, distribution and services, clouds can be classified among :

- Private, which is restricted for management and usage of predefined users.

- Public or Hosted, which is open to the public, generally usually charged on a pay-per-use.

- Community, which is available to specific group of people or community in order to share resources and services.

- Hybrid, which is a combination of the other three types.

And so, the applicability of Cloud Computing has been studied. Luiz Bittencourt et.al [9] depict how the expansion of Internet of Things (IoT), has affecting the way to use Cloud Computing, storing, processing and producing information and knowledge as a result. It also discuss how in one hand, the wide adoption of cloud computing is a consequence of a fast time-to-market for many types of applications due to the paradigm's flexibility and reduced or null initial capital expenditures, on the other hand, this same wide adoption has exposed some limitations of the paradigm in fulfilling all requirements of some classes of applications, such as real-time low latency, and mobile applications. And examples, due to the centralized cloud data centers are often physically and/or logically distant from the cloud client, the communication and data transfers traverses multiple hops, which introduces delays and consumes network bandwidth of edge and core networks. As a combination of the ability of run small, localized applications at the edge with the high-capacity from the cloud, it presents the fog computing as emerged paradigm that can support heterogeneous requirements of small and large applications through multiple layers of a computational infrastructure that combines resources from the edge of the network as well as from the cloud.

in addition, Y. Mao et.al [24] present how Mobile devices tends to growth in terms of usability and processing of data, implicating the decentralization from the Cloud's presence. This survey says that the last decade has seen Cloud Computing emerging as a recent paradigm of computing such that a vision is the centralization of computing, storage and network management in the Clouds, referring to data centers, backbone IP networks and cellular core networks. But, in recent years, it has been changed due to the Clouds being increasingly moving towards the network edges. Y. Mao et.al present the estimation that tens of billions of Edge devices will be deployed in the near future, and their processor speeds are growing exponentially, following Moore's Law. Harvesting the vast amount of the idle computation power and storage space distributed at the network edges can yield sufficient capacities for performing computation-intensive and latency-critical tasks at mobile devices. The same survet presents the Mobile Edge Computing (MEC) as a computation provider at mobile devices considering the proximate access, that is widely agreed to be a key technology for realizing various visions for next-generation Internet, such as Tactile Internet (with millisecond-scale reaction time) and Internet of Things (IoT). Also, Y. Mao et.al say :

- it shows implications from the different techniques of implementation of MEC, which are network functions virtualization (NFV), information-centric networks (ICN) and software-defined networks (SDN).

- it presents the Fog Computing as a propose of Cisco as a generalized form of MEC where the definition of edge devices gets broader, Fog Computing and Networking are overlapping the terminologies with MEC.

2.2 Resource Allocation Techniques and Metrics

Some of the works referenced in the previous section also present the challenges for resource allocation from the Cloud, Grid and Edge Computing. Hence, in this section will be presented this solutions and techniques.

Huang et.al [21] affirm that to make appropriate decisions when allocating hardware resources to the tasks and dispatching the computing tasks to resource pool has become the main issue in cloud computing. As cloud computing has its own features, the resource allocation policies and scheduling algorithms for the other computing technologies are unable to work under this conditions. For that reason there is not an uniform standard for job scheduling in cloud and then it is an important component in this context.

S. M. Parikh [31] points that the management of flexible resources allocation is a problem emerged in this context, due to heterogeneity in hardware capabilities, workload estimation and a variety of services, also as the the maximization of the profit for cloud providers and the minimization of cost for cloud consumers.

According to Hameed Hussain et.al [22] the resource management mechanism determines the efficiency of the used resources and guarantees the Quality of Service (QoS) provided to the users. Therefore, the resource allocation mechanisms are considered a central theme in HPCs. QoS resource management and scheduling algorithms are capable of optimally assigning resources in ideal situation or near-optimally assigning resources in actual situation, taking into account the task characteristics and QoS requirements. In addition it presents common attributes among the HPC categories, such as size, network type, and coupling.

In Grid platforms as Qureshi, Muhammad Bilal et.al [33] present, a Grid resource can be defined as an entity that needs to carry out an operation by an application such that each application in Grid environment competes for various resources according to application needs. This way resource allocation, denoted by RA, mechanisms play an important role in allocating the most appropriate resources to applications. The mechanisms perform the allocation of tasks to the resources in order to ensure QoS to the application according to the user requirements. RA mechanisms provide two basic Grid services:

- Resource monitoring, which regularly monitors resource performance, capability, usage and future reservations, including processors, disks, memories,and channel bandwidths.

- Resource scheduling, which retrieves the information from a) and decides on the allocation of the application to the underlying resources. There are several goals to conduct the RA process as reduce makespan, power minimization and energy efficiency improvement, reduction of task completion time or the amount of data transfer, among others.

For the other side, Feitelson, Dror G. [17] says that the root cause for convergence problems is variability in the workloads. Therefore, it characterizes the variability in the runtimes and arrives of workloads observed on different systems, and in models based on them. The first metric dealt with is the response time. It defines “response time” to mean the total wall clock time from the instant at which the job is submitted to the system, until it finishes its run. This can be divided into two components: the running time, denoted by Tr, during which the job is actually running in parallel on multiple processing nodes, and the waiting time, denoted by Tw , in which it is waiting to be scheduled or for some event such as I/O. The waiting time itself can also be used as a metric, based on the assumption that Tr does not depend on the

scheduling. Obviously, a lower bound on the response time of a given job is its running time. As the runtimes of jobs have a very large variance, so must the response time. It has therefore been suggested that a better metric may be the slowdown (also called "expansion factor"), which is the response time normalized by the running time: $slowdown = (Tw + Tr)/Tr$. Thus if a job takes twice as long to run due to system load, it suffers from a slowdown factor of 2, etc. This is expected to reduce the extreme values associated with very long jobs, because even if a week-long job is delayed for a whole year the slowdown is only a factor of 50. Moreover, slowdown is widely perceived as better matching user expectations that a job's response time will be proportional to its running time. It affirms that the slowdown metric is that it over-emphasizes the importance of very short jobs. For example, a job taking 100 ms that is delayed for 10 minutes suffers from a slowdown of 6000, whereas a 10-second job delayed by the same 10 minutes has a slowdown of only 60. To avoid such effects, Feitelson et al. have suggested the "bounded-slowdown" metric. The difference is that for short jobs, this measures the slowdown relative to some "interactive threshold", rather than relative to the actual runtime. Denoting this threshold by , the definition is $bounded - slowdown = max\{(Tw + Tr)/max\{Tr, \}, 1\}$

In addition, the Aida, Kento [4] deals with the investigation of the effect of the job size on the scheduling performances, it characterizes and performs experiments showing how does the processor utilization and the bounded slowdown are affected in that context. In details, the processor utilization is the percentage that processors are busy over entire simulation. The slowdown ratio (SR), shows normalized data for mean response time, and it is derived by the following formula. For instance, Aida, Kento supposes that 10000 jobs were executed in an experiment. The mean response time of these 10000 jobs was 5 hours, and their mean execution time on processors was 2 hours. Then, the slowdown ratio is 2.5. Instead other many previous performance evaluation works that assumed that characteristics of parallel jobs, or a parallel workload, followed a simple mathematical model, Aida, Kento follows recent analysis of real workload logs, which are collected from many large-scale parallel computers in production use, shows that a real parallel workload has more complicated characteristics.

2.3 Applicability, Load Balance, Energy Consumption Reduction and Other

Different techniques for several goals have been applied in the context of Cloud and Edge Computing. The usage of containers is one technique presented by C. Pahl and B. Lee [30] that introduce the Cloud computing as a centralized, large-scale data centres to a more distributed multi-cloud setting comprised of a network of larger and smaller virtualized infrastructure run-time nodes, also referred to as edge clouds, edge computing or fog computing. It is focused on the virtualization as a form to reach the network and allow Internet-of Things (IoT) infrastructures to be integrated. As a challenge resulting from distribution, it affirms the necessity of more lightweight solutions than the current virtual machine (VM)-based virtualisation technology. Virtual machines (VMs) have been at the core of the compute infrastructure layer providing virtualized operating systems. It investigates containers, which are a lightweight virtualisation concept, i.e., less resource and time consuming. VMs and containers are both virtualisation techniques, but solve different problems. Containers are a solution for more interoperable application packaging in the cloud and should therefore address the PaaS concerns.

Load balancing algorithm is another example, M. Randles et.al [34] identify it as major

concern to allow Cloud computing to scale up to increasing demands. It presents three potentially viable methods for load balancing in large scale Cloud systems. The first one is a nature-inspired algorithm may be used for self-organization, achieving global load balancing via local server actions. The second one by a self-organization that can be engineered based on random sampling of the system domain, giving a balanced load across all system nodes. The third one, by a restructure to optimize job assignment at the servers.

Since the execution of HPC jobs produce a huge energy consumption, one other target that has been studied is how to reduce this energy consumption. Jie Meng et.al [26] presents that has been reported the worldwide data center electricity consumption increased by 56% from 2005 to 2010, which accounted for 1.3% of the total electricity use. A recent review shows that for every dollar spent on power of data center computing equipments, another dollar is spent on data center cooling infrastructures, which translates to an energy cost reaching up to millions of dollars and cooling costs reaching close to half of the overall energy cost. Thus, it manages simulations in order to study cooling and energy efficiency in this context.

The Batsim/ SimGrid toolkit also includes a plugin to keep track of temperature for similar reasons [35, 19]. These papers show how the energy is modeled and how it could be used to achieve such target, which will be discussed a lit bit more in next sections.

in addition, the Qarnot Computing proposal is direct related with this context, which will discussed in the next sections, but could be find detailed information in the paper [29]

2.4 Motivation, Companies Usage and Recent Statistics

The work [36] presents in details how Cloud computing has been used and how it has been requiring Edge computing as a recent paradigm. It takes in account the aspect and impact commercial from these computing platforms providing a very good panoramic view of the context, it affirms that nascent technologies and applications for mobile computing and the Internet of Things (IoT) are driving computing toward dispersion. For them Edge computing is a recent paradigm in which substantial computing and storage resources—variously referred to as cloudlets, micro datacenters, or fog nodes are placed at the Internet's edge in close proximity to mobile devices or sensors. It shows that industry investment and research interest in edge computing have grown dramatically in recent years. Nokia and IBM jointly introduced the Radio Applications Cloud Server (RACS), an edge computing platform for 4G/LTE networks, in early 2013. It affirms that the following year, a mobile edge computing standardization effort began under the auspices of the European Telecommunications Standards Institute (ETSI). The Open Edge Computing initiative (OEC; openedgecomputing.org) was launched in June 2015 by Vodafone, Intel, and Huawei in partnership with Carnegie Mellon University (CMU) and expanded a year later to include Verizon, Deutsche Telekom, T-Mobile, Nokia, and Crown Castle. This collaboration includes creation of a Living Edge Lab in Pittsburgh, Pennsylvania, to gain hands-on experience with a live deployment of proof-of-concept cloudlet-based applications. Organized by the telecom industry, the first Mobile Edge Computing Congress (tmt.knect365.com/mobile-edge-computing) convened in London in September 2015 and again in Munich a year later. The Open Fog Consortium (www.openfogconsortium.org) was created by Cisco, Microsoft, Intel, Dell, and ARM in partnership with Princeton University in November 2015, and has since expanded to include many other companies. The First IEEE/ ACM

Symposium on Edge Computing (conferences.computer.org/SEC) was held in October 2016 in Washington, DC.

Is possible to find examples of software as a service (SaaS) instances, such as Google Apps, Twitter, Facebook, and Flickr, have been widely used in our daily life [38], [39]. Moreover, scalable infrastructures as well as processing engines developed to support cloud service are also significantly influencing the way of running business, for instance, Google File System, MapReduce, Apache Hadoop, Apache Spark, and so on. In addition, it affirms with recent statistics that with IoT, we will arrive in the post-cloud era, where there will be a large quality of data generated by things that are immersed in our daily life, and a lot of applications will also be deployed at the edge to consume these data. By 2019, data produced by people, machines, and things will reach 500 zettabytes, as estimated by Cisco Global Cloud Index, however, the global data center IP traffic will only reach 10.4 zettabytes by that time. By 2019, 45% of IoT-created data will be stored, processed, analyzed, and acted upon close to, or at the edge of, the network. Finally, they raise the following issues: *Why Do We Need Edge Computing ?*

- Push from Cloud services: putting all the computing tasks on the cloud has been proved to be an efficient way for data processing since the computing power on the cloud outclasses the capability of the things at the edge. However, compared to the fast developing data processing speed, the bandwidth of the network has come to a standstill. With the growing quantity of data generated at the edge, speed of data transportation is becoming the bottleneck for the cloud-based computing paradigm. It examples, about 5 Gigabyte data will be generated by a Boeing 787 every second, but the bandwidth between the airplane and either satellite or base station on the ground is not large enough for data transmission. It considers an autonomous vehicle as another example, one Gigabyte data will be generated by the car every second and it requires real-time processing for the vehicle to make correct decisions. If all the data needs to be sent to the cloud for processing, the response time would be too long. Not to mention that current network bandwidth and reliability would be challenged for its capability of supporting a large number of vehicles in one area. In this case, the data needs to be processed at the edge for shorter response time.

- Pull From IoT: almost all kinds of electrical devices will become part of IoT, and they will play the role of data producers as well as consumers, such as air quality sensors, LED bars, streetlights and even an Internet-connected microwave oven. It is safe to infer that the number of things at the edge of the network will develop to more than billions in a few years. Thus, raw data produced by them will be enormous, making conventional cloud computing not efficient enough to handle all these data. This means most of the data produced by IoT will never be transmitted to the cloud, instead it will be consumed at the edge of the network.

- Change from data consumer to producer: in the cloud computing paradigm, the end devices at the edge usually play as data consumer, for example, watching a YouTube video on your smart phone. However, people are also producing data nowadays from their mobile devices. The change from data consumer to data producer/consumer requires more function placement at the edge. For example, it is very normal that people today take photos or do video recording then share the data through a cloud service such as YouTube, Facebook, Twitter, or Instagram. Moreover, every single minute, YouTube

users upload 72 h of new video content; Facebook users share nearly 2.5 million pieces of content; Twitter users tweet nearly 300 000 times; Instagram users post nearly 220 000 new photos. However, the image or video clip could be fairly large and it would occupy a lot of bandwidth for uploading. In this case, the video clip should be demised and adjusted to suitable resolution at the edge before uploading to cloud. Another example would be wearable health devices. Since the physical data collected by the things at the edge of the net- work is usually private, processing the data at the edge could protect user privacy better than uploading raw data to cloud.

Also, *what are the benefits of Edge Computing?*

- In edge computing we want to put the computing at the proximity of data sources. This have several benefits com- pared to traditional cloud-based computing paradigm. Here we use several early results from the community to demonstrate the potential benefits. Researchers built a proof-of-concept platform to run face recognition application in, and the response time is reduced from 900 to 169 ms by moving com- putation from cloud to the edge. Moreover, the energy consumption could also be reduced by 30%–40% by cloudlet offload- ing. clonecloud in combine partitioning, migration with merging, and on-demand instantiation of partitioning between mobile and the cloud, and their prototype could reduce 20× running time and energy for tested applications.

2.5 Related Simulation Tools

We described in this thesis a novel simulation tool for easily designing and testing scheduling strategies on edge computing platforms. It was motivated the huge effort of building a new simulator using adequate tools for modeling the processing and memory units and the network topology.

We discussed briefly below the main competitors and argument for this simulator. Some simulators have constraints that would prevent us to correctly simulate a platform such as the *Qarnot*'s one. For example, EmuFog[25] does not support hierarchical fog infrastructures, whereas *Qarnot* infrastructure is inherently hierarchical. Other simulators such as iFogSim[18], EdgeCloudSim[41] and IOTsim[44], are simulation frameworks that enable to simulate fog computing infrastructures and execute simulated applications on top of it. The two closest simulators to the presented one is a) the CloudSim, widely used to validate algorithms and applications in different scientific publications, howerver is based on a top-down viewpoint of cloud environments. And b) this one related to other very close work on the literature [26] where are implemented evaluation models and allocation optimization methods in SST, the Structural Simulation Toolkit. The SST is an architectural simulation framework designed to assist in the design, evaluation, and optimization of HPC architectures and applications. It is developed by Sandia National Laboratories to evaluate the performance of computer systems ranging from small-scale single-chip processors to large-scale parallel computing architectures. It was used for evaluating an optimization algorithm managing real-world parallel workloads, as well as the implementations of job scheduler and allocation algorithms in SST.

That is, to the best of our knowledge, there are no articles that properly validate the different models it relies on. On the contrary, the presented one is built on top of SimGrid, which has been validated in many publications [2] and allows finer-grained simulations, as explained in Section 3.1.

— 3 —

A Dedicated Scheduling Simulator for Edge Platforms

We chose to Batsim/SimGrid instead of available fog/edge simulators [18, 41, 44] for several reasons:

- Batsim has been specially 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.

Released in 2017, Batsim [15] delivers a high-level API to facilitate the development of batch scheduling algorithms that can be then simulated on top of SimGrid [12], the well-proven simulator toolkit for distributed infrastructures. Thus, is possible to rely on high-level tools that have been proposed and already validated. Some parts have been customized to reflect the edge specifics, especially the decision processes as a goal of this thesis, and the others have been developed by the workgroup at the same time. Both will be explained in the following.

3.1 Operational Components

In this section is discussed the role of the different components, namely SimGrid, Batsim, the *decision process* and their interactions.

3.1.1 SimGrid

SimGrid [12] 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.

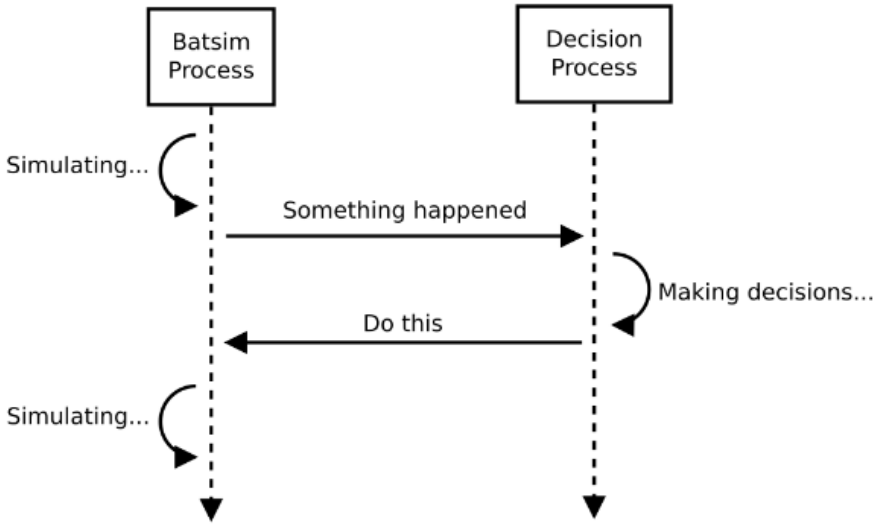


Figure 3.1 – Batsim and decisions maker interaction.

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 the easy simulation of complex systems such as fog/edge infrastructures.

3.1.2 Batsim and the Decision Process

Batsim [15] 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 behavior 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.

Figure 3.1 illustrates the interaction between Batsim and the decision process. It 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 the decision process, which provides functions to ease the communication with Batsim.

More details on Batsim and SimGrid mechanisms can be found on Chapter 4 of Millian Poquet's manuscript [32].

3.2 Extensions

There are a couple of extensions that have been developed to deal with edge challenges. In this section, is presented 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.

3.2.1 Batsim/SimGrid Plug-in

One of the strengths of SimGrid is its plug-in architecture. For instance, one plug-in of interest, has 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 [20].

As will be discussed later in the document, was leveraged this extension in the *Qarnot* case study. Concretely, was 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.

3.2.2 External Events Injector

To simulate the execution of a fog/edge infrastructure, which by essence is subject to very frequent unexpected or unpredictable changes, this simulator offers the opportunity to inject external events on demand. Those events impact the behavior 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 (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.

An event is represented as a JSON object that contains two mandatory fields: a *timestamp*, which 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 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 the 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.

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

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

3.2.3 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, 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.

— 4 —
Case study: the Qarnot Computing platform

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

4.1 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 symmetric 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.

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.

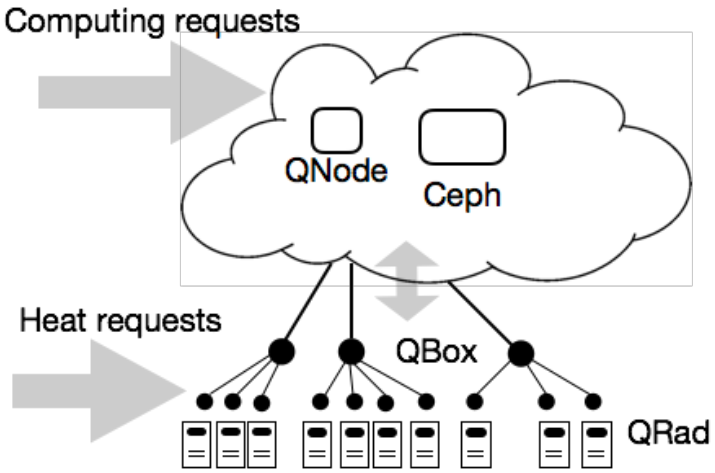


Figure 4.1 – Scheme of the Qarnot platform.

4.2 Platform Terminology

The job and resource manager of the *Qarnot* platform, named *Q.ware*, is based on a hierarchy of 3 levels as shown in Figure 4.1: 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 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 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 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 increase the heterogeneity challenges of and edge infrastructure: the computation resource does not simply appear or disappear but also varies according to the heating needs.

4.3 Current Workflow

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 many instances of the QTasks ordered by priority on QBoxes, depending on the number 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 instances.

Notice that, at all times, a *FrequencyRegulator* 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.

Figure 4.2 summarizes the execution flow of a QTask within the *Qarnot* platform.

As said before, QTasks to be executed on this platform are submitted to the QNode and all the data dependencies are uploaded to the CEPH (steps 01 and 02). To be executed, these QTasks, along with their data dependencies, have to be sent to the QBoxes and then scheduled onto QMobos. Every 30 seconds, the QBoxes send information about the state of their disk, QRads and QMobos to the QNode (steps 03 through 05), in particular, the number of QMobos available for computation and the free storage space are reported.

A first scheduling process is made at the QNode-level to dispatch instances of the QTasks to the QBoxes (steps 06 and 07). The QNode tries to dispatch, for each QTask taken by priority, as many instances as possible onto QBoxes, with respect to the number of available QMobos and storage space left on the QBox disks.

Upon receiving instances of a QTask, the QBox will reserve for each instance a QMobo from the warmest QRad for the case of low priority QTask, and a QMobo from the coolest QRad in the case of high priority (step 08). This distinction is made to keep more QMobos available in case high priority QTasks are sent to the QBox in the near future. The QBox then checks whether the Docker image and other data dependencies for these instances are on disk and fetches any missing data from the CEPH (steps 09 and 10).

Once all data transfers are completed for this QTask, the reserved QMobos are rebooted and the instances are started (steps 11 through 14). When the instance completes, its output is uploaded to the CEPH and the QNode is notified of the instance completion (steps 15 through 17). Finally, the queue of QTasks is updated and if an instance of the same QTask can be directly dispatched, it is sent to the QBox and the execution starts immediately, without rebooting the QMobo (steps 18 and 19).

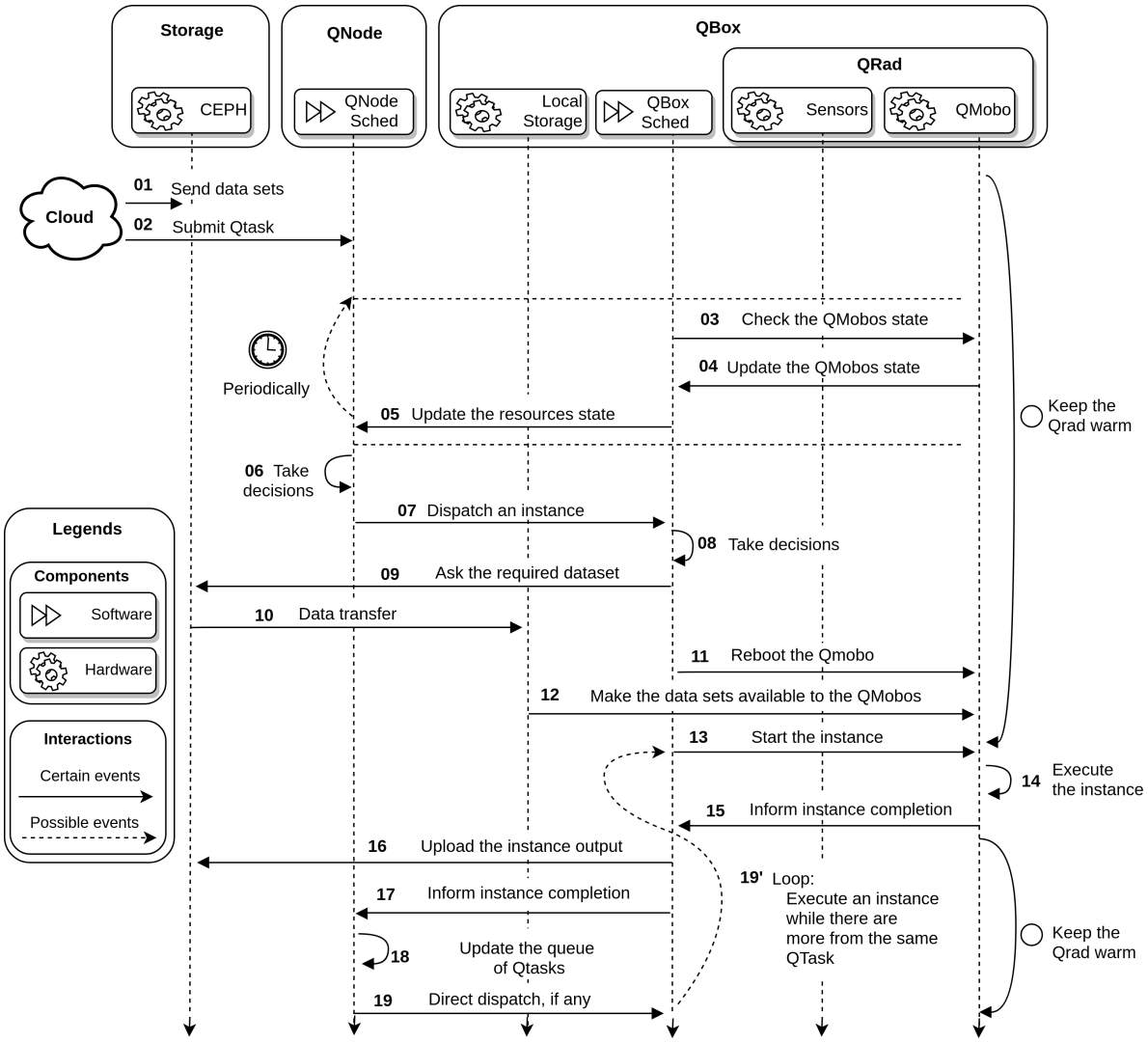


Figure 4.2 – Execution flow of a QTask on the Qarnot platform.

— 5 —
Simulated Platform

In this section is presented how was modeled the *Qarnot* platform and is explained how it was instantiated the inputs that were required for the simulation. Then, it is described the content of the files given as input of the simulation and how they are generated.

5.1 Qarnot to Batsim/SimGrid Abstractions

Figure 5.1 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 SimGrid zone, as well as QRads of the same QBox, and all QBoxes of the QNode. The management of storage spaces is done by adding special hosts that 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.

5.2 Workflow

In order to simulate the behavior presented in Section 4.1 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. Through the platform translation as Figure 5.1, the Qarnot platform has been simulated as Figure 5.2.This one illustrates in detail how the abstraction between the Qarnot and the simulated platform is performed, where is possible to see fewer components for the same workflow since Batsim is in charge of all physical and execution process.

The simulated platform considers 3 main classes: the QNodeSched, QBoxSched and StorageController. Each one can communicate with the Batsim process delivering or receiving information by messages.

We illustrate the simulated platform workflow in Figure 5.2. First, Batsim receives as input and loads the events, platform, workload and the data sets description (steps 01 and 02). Then, the QNodeSched will be notified to take some decision (step 03). First, it should know which

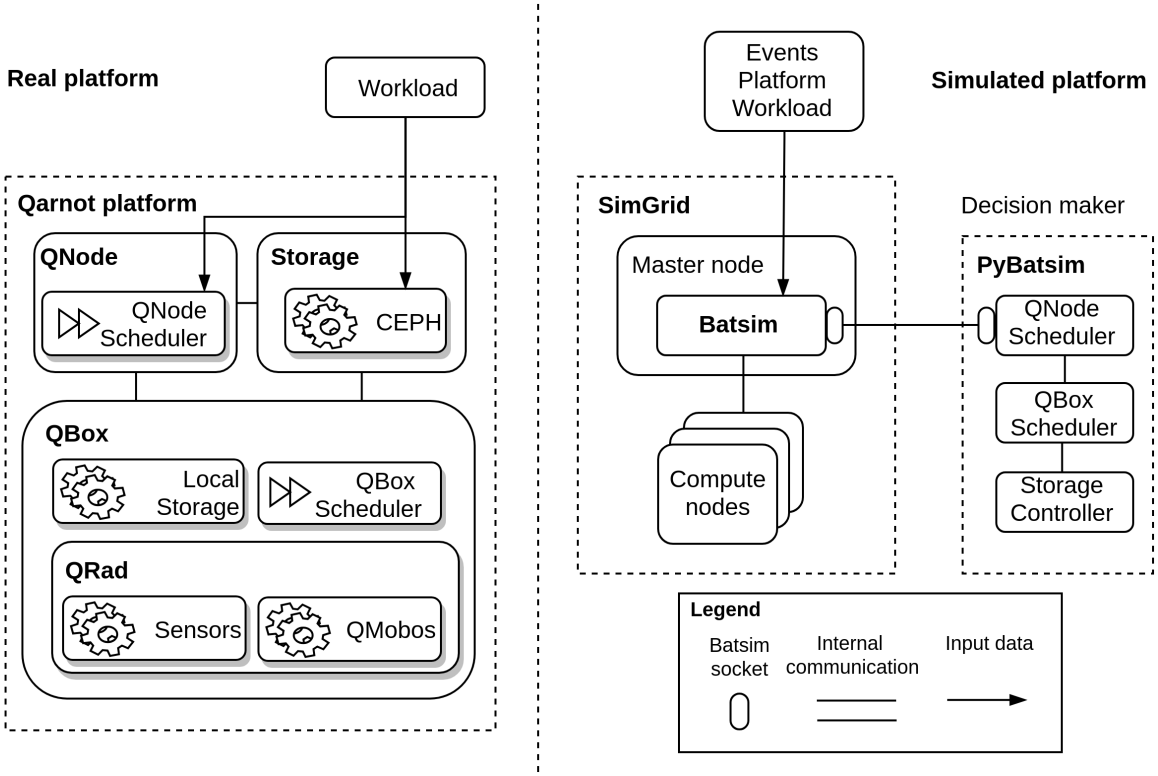


Figure 5.1 – Comparison between the real and simulated *Qarnot* platform.

resources are available. For that, every 30 seconds (defined as default), the QNodeSched ask information about the state of the disk, QRads and QMobos of the QBoxes (steps 03 and 04), in particular, the number of QMobos available for computation and the free storage space are reported.

A first scheduling process is made at the QNode-level to dispatch instances of the QTasks to the QBoxes (steps 05 and 06). The QNode tries to dispatch, for each QTask taken by priority, as many instances as possible onto QBoxes, with respect to the number of available QMobos and storage space left on the QBox disks.

Upon receiving instances of a QTask, the QBox will reserve for each instance a QMobo from the warmest QRad for the case of low priority QTask, and a QMobo from the coolest QRad in the case of high priority (step 08). This distinction is made to keep more QMobos available in case high priority QTasks are sent to the QBox in the near future. The QBox then checks whether the Docker image and other data dependencies for these instances are on disk and fetches any missing data from the CEPH (steps 08 and 09).

Once all data transfers are completed for this QTask, the QBoxSched will be informed by the StorageController and then will allow Batsim to simulate the instance execution, which will notify the QNodeSched whenever the instance completes (steps 11 through 14).

Finally, the queue of QTasks is updated and if an instance of the same QTask can be directly dispatched, it is sent to the QBox and the execution starts immediately, without rebooting the QMobo (steps 15 through 16).

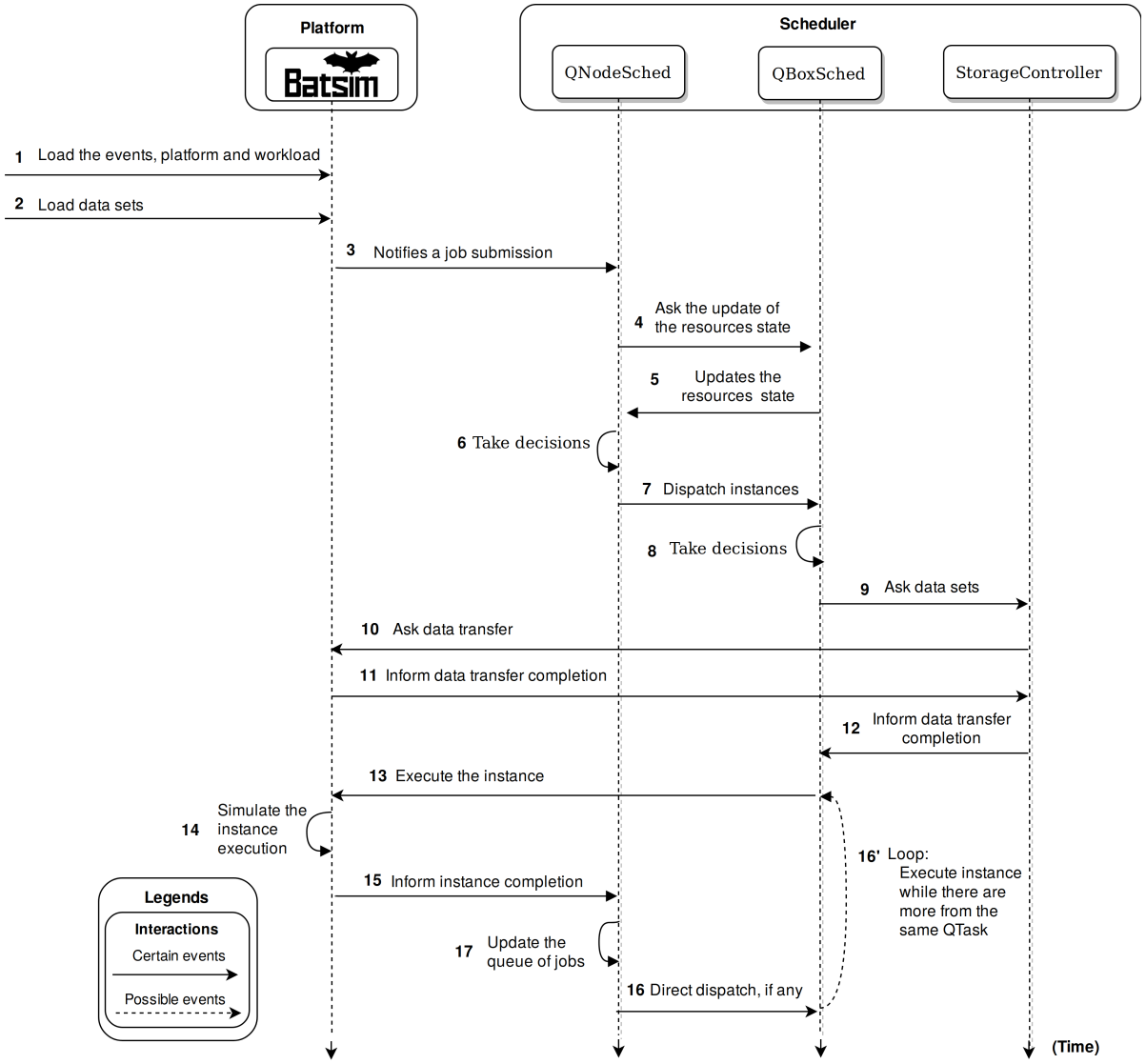


Figure 5.2 – Simulated platform on SimGrid/ Batsim.

5.3 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.

5.3.1 Platform Description

The definition of the platform is an XML file readable by SimGrid. The file describes the whole platform to simulate, within 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,
- A list of QRads with for each:
  - The id
  - A list of QMobos with for each:
    * The id,
    * The list of speeds and corresponding power usage,
    * The coefficients required for the temperature plug-in.

5.3.2 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,
- The job profile to use.

Profile descriptions represent how a job should be simulated, plus other specific information, and contain:

- The type of job to simulate,
- The number of flops to compute,
- The job priority,
- The list of data-sets required as inputs.

In addition, each *Qarnot* instance is represented by a Batsim job with a specific job profile and requiring a single computing resource. Also, each job extracted provides the following information, not used by Batsim, but useful for some analysis that will be presented in further sections:

- The time when the job was started in the real platform,
- The time when the job finished in the real platform,
- The id of the real resource where the QTask was executed,
- The speed of execution from the real platform.

5.3.3 Data Sets Description

The list of data-sets is also described as a list of JSON objects, with one per line in the file is read by the *decision process* and fed to the Storage Controller. Each data-set object is represented by two fields:

- The id
- The size in bytes of the data-set.

5.3.4 External Events Description

As discussed in Section 3.2.2, each event is timestamped and is described as a JSON object:

- *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 3.2.1).

In addition there is 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. It is useful if one intends to simulate a specific period of time, for example, the behavior of a platform for one week.

— 6 —
Job Allocation

As presented in Chapter 2, the HPC job allocation problem has been studied since a long time ago, and the main idea is to fit, by some convenient metric, HPC jobs to be executed as soon as possible onto resources. The metrics and weights taken into account for the decision processes could change as the proposals of the system, but in general, from a queue of jobs, allocation decisions among resources should be taken.

As the goal of this thesis, several scheduling policies were implemented in order to compare their performances. In this section will be described the challenges present in its use-case and all algorithms for the implemented policies.

6.1 Scheduling Challenges

Was described in Section 4.1 the Qarnot proposal, which is naturally a multi-objective problem, because there are at least three viewpoints, the first one from customers that want to compute (HPC customers), the second one from customers that want to be heated (host) and the third one from the middleware.

The viewpoint of the HPC customers is what is found in classical distributed scheduling systems: the goal is to get the results of submitted jobs as soon as possible. The viewpoint of the hosts completely differs from what is found in classical scheduling theory since the Qarnot foundation. Finally, the middleware viewpoint is specifically related to a goal of the Qarnot computing business model, which is the one of reducing the energy consumption in the processing of the jobs, related to the Qarnot business model since the company re-funds the electricity bill of the hosts.

Following the Q.Ware infrastructure, the decision processes are taken in two-levels, through the QNodes and the QBoxes. The scheduling at QNode-level can be viewed as an assignment step. The idea is to dispatch in priority tasks to QBoxes having QRads which need to heat the most. The scheduling at the QBox-level is in charge to select the best QRad based on its QMobos, also download the required data-sets and report periodically the status of its resources.

From a QNode point of view, the notion of temperature and heating needs is unknown and hidden behind other information sent periodically by the QBoxes, as detailed below. Upon the arrival of a task, the QNode knows the following information:

- The number n of instances composing the task.
- The priority w of the task.

- The list of data sets $\{D_1 \cdots D_k\}$ which the task depends on.

The priorities are defined as high, low and background, such that QTasks with high priority can preempt low and background others, QTasks with low priority can preempt background others and QTasks with background priority can not preempt any other. This last one is not a real HPC or IoT job submitted on the platform. Named *burn_job*, this Qtask is a fake job created locally when some user requires heating and there are no jobs to be sent to that QRad. This way, this job is useful just to provide heating, been possible to be preempted anytime whenever a real job arrives on that QRad.

Moreover, from periodic reports from the QBoxes, the QNode knows the number of available QRads for each type of priority, which is critical to achieving a good quality of service regarding the priority of the tasks. For example, if there are not enough available QRads to start a high priority task, some lower priority tasks being executed must be preempted to free resources for that high priority task. Hence is necessary to have different values for the available QRads, one for each different class of task priority.

The last source of information to help the QNode dispatch tasks to QBoxes is provided by the storage controller, which is in charge of managing the data sets available in the centralized storage and the QBox disks, as well as their movements. It provides the list of QBoxes already having the data sets required by that task. This way, the Algorithm 1, 2, 3, 4, 5 and 6 were developed.

6.2 Standard Schedulers

The standard scheduler for both levels was based on the current Qarnot QNodes and QBoxes policies.

6.2.1 QNode Scheduler

The QNode scheduler is in charge of manage the queue of QTasks and dispatch to the QBoxes that require heating, taking into account the priority of the QTasks. It is the central decision maker and has a global view of the process, receiving information about the available resources from the QBoxes and QTasks from submissions of HPC jobs or from IoT devices. This scheduler was implemented as Algorithm 1.

6.2.2 QBox Scheduler

The QBox scheduler is in charge to require the data sets to the Storage Controller, dispatch jobs to the QRads and start the execution of the QTasks in the QMobos whenever the data sets are ready.

Algorithm 2 aims to execute an instance of high priority on the coolest QRad available for that, if possible without preempting low instance. Execute an instance with background/low priority on the warmest QRad available for that.

Algorithm 1 QNode scheduler: dispatching instances onto QBoxes - Standard version

available_mobos_list ← List of available QMobos among all QBoxes
qtask_queue ← The list of QTasks to be dispatched
if *qtask_queue*$\emptyset$ **then**
    return
else
    Sort the *qtask_queue* by 1) decreasing *priority*; 2) increasing $nb\_of\_running\_instances$
    for $qtask \in qtask\_queue$ **do**
        $nb\_instances\_left$ ← Number of instances of *qtask* waiting to be dispatched
        if $nb\_instances\_left > 0$ **then**
            mobos_list ← List of QMobos from *available_mobos_list* with BKGD priority
            Dispatch as many instances as possible on QMobos from *mobos_list*
            if $nb\_instances\_left > 0$ **and** $qtask.priority\_group >$ BKGD **then**
                # There are more instances to dispatch and the qtask is either LOW or HIGH priority.
                mobos_list ← List of QMobos from *available_mobos_list* with LOW priority
                Dispatch as many instances as possible on QMobos from *mobos_list*
                if $nb\_instances\_left > 0$ **and** $qtask.priority\_group >$ LOW **then**
                    # There are more instances to dispatch and the qtask has HIGH priority.
                    mobos_list ← List of QMobos from *available_mobos_list* with HIGH priority
                    Dispatch as many instances as possible on QMobos from *mobos_list*

Algorithm 2 QBox scheduler: dispatching instances onto QRads

waiting_instances ← List of instances waiting to be scheduled on this QBox, sorted by priority
for $qtask \in waiting\_instances$ **do**
    Ask for the transfer of data-sets from the *CEPH* to the *QBox* disk.
    if $qtask.priority$ HIGH **then**
        # Find coolest QRad which is not running LOW instance
        qmobo_list ← List of QMobos which is possible to run HIGH priority Qtasks
        qrad_list ← List of QRads by decreasing temperature
        Run as many instances as possible in $Qmobos \in qmobos\_list$ which are from $Qrads \in qrad\_list$
    else
        # Find warmest QRad among the availLow and availBkgd
        qmobo_list ← List of QMobos which is possible to run LOW priority Qtasks
        qrad_list ← List of QRads by decreasing temperature
        Run as many instances as possible in $Qmobos \in qmobos\_list$ which are from $Qrads \in qrad\_list$
if $waiting\_instances \neq 0$ **then**
    # Some *qtaks* could not be executed.
    Reject the *waiting_instances* to the QNode Scheduler

6.3 QNode Schedulers Variants

Based on the standard QNode scheduler we built the other four variants. This way we compared its performances in order to describe which one fits better for the use-cases.

6.3.1 Locality Based Scheduler

The *LocalityBased* scheduler gives priority to the QBoxes already having in disk all data dependencies of the QTask to be dispatched. This first variant, Algorithm 3, aims at avoiding useless data transfers if some QBoxes already have the required data dependencies of a given QTask.

Algorithm 3 QNode scheduler: dispatching instances onto QBoxes - Locality based version

```
    available_mobos_list ← List of available QMobos among all QBoxes
 2: qtask_queue ← The list of QTasks to be dispatched
    if qtask_queue∅ then
 4:   return
    else
 6:   Sort the qtask_queue by 1) decreasing priority; 2) increasing nb_of_running_instances
      for qtask ∈ qtask_queue do
 8:     nb_instances_left ← Number of instances to be dispatched by the qtask
        if nb_instances_left > 0 then
10:       list_qboxes ← List of QBoxes with the data sets required by the qtask.
          mobos_list ← List of QMobos from available_mobos_list with BKGD priority.
12:       mobos_list ← mobos_list filtered by QMobos from QBoxes in list_qboxes.
          for mobo ∈ mobos_list do
14:         Dispatch as many instances as possible on mobo.
          if nb_instances_left > 0 and qtask.priority_group > BKGD then
16:         # There are more instances to dispatch and the qtask is either LOW or HIGH
            priority.
            mobos_list ← List of QMobos from available_mobos_list with LOW priority
18:         mobos_list ← mobos_list filtered by QMobos from QBoxes in list_qboxes.
            for mobo ∈ mobos_list do
20:           Dispatch as many instances as possible on mobo.
            if nb_instances_left > 0 and qtask.priority_group > LOW then
22:           # There are more instances to dispatch and the qtask has HIGH priority.
              mobos_list ← List of QMobos from available_mobos_list with HIGH priority
24:           mobos_list ← mobos_list filtered by QMobos from QBoxes in list_qboxes.
              for mobo ∈ mobos_list do
26:             Dispatch as many instances as possible on mobo.
    Apply Algorithm Algorithm 1
```

6.3.2 Full Replicate Scheduler

Namely *FullReplicate*, it replicates all data dependencies of a QTask on all QBox disks before that QTask arrives in the system. This variant, Algorithm 4, aims at visualizing the

behaviors of the scheduling policy without any impact of the data movements.

Algorithm 4 QNode scheduler: dispatching instances onto QBoxes - Full Replicate version

Replicate all data sets to all QBoxes.
Apply Algorithm Algorithm 1

6.3.3 3-Replicated and 10-Replicated Schedulers

Finally, the schedulers *Replicate3* (Algorithm 5) and *Replicate10* (Algorithm 6), 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 LocalityBased dispatching policy.

As the $LocalityBasedScheduler$ transfers new data sets whenever it is required by some QBox and the $FullReplicateScheduler$ transfers all data sets to all QBoxes whenever a qtask arrives in the QNode, the 3-10$Replicate$ schedulers are two trade-offs between 0% and 100% of data replication before dispatching. They aim at reducing the waiting time of the instances of QTasks by providing more QBox candidates for the LocalityBased dispatcher.

Algorithm 5 QNode scheduler: dispatching instances onto QBoxes - Replicate3 version

Whenever a QTask is submitted:
  $qbox\_list \leftarrow$ The list of the 3 QBoxes with most empty disks.
3: Replicate all data sets on QBoxes from $qbox\_list$.
Apply Algorithm Algorithm 3

Algorithm 6 QNode scheduler: dispatching instances onto QBoxes - Replicate10 version

Whenever a QTask is submitted:
  $qbox\_list \leftarrow$ The list of the 10 QBoxes with most empty disks.
  Replicate all data sets on QBoxes from $qbox\_list$.
4: Apply Algorithm Algorithm 3

— 7 —

Experiments, Discussions and Results

By the logs extracted detailed in Section 5.3, we were able to extract workloads for specific periods of time. To compare the results of simulations we utilized workloads of one and three days, one and two weeks. As the one and three days show less information and the two weeks show a lot of it, this section will be presented results regarding workloads with the size of one week. Due to recent modifications in the extractor, we had less than two months of data available. To characterize one full month, we will present four workloads, with each one stated from 03, 10, 17 and 24 of May, denoted respectively as 1w_03, 1w_10, 1w_17 and 1w_24. In addition, the platform simulated was composed by approximately 3390 QMobos, from 669 QRads, managed by 20 QBoxes.

7.1 Job's Processing Time

By the extracted logs, the processing time distribution for each workload was computed, such that, for each QTask in each workload: $processing\_time = real\_finish\_time - real\_start\_time$. In other words, was computed a list of processing time for each instance of a workload, then a distribution could be done and is presented in Table 7.1.

Table 7.1 – Processing time distribution for different weeks of workload

Statistics	**1w_03**	**1w_10**	**1w_17**	**1w_24**
Count	7350	5989	5497	8850
Mean (s)	465.96	582.25	480.21	403.93
Std (s)	817.18	2400.22	2268.20	1723.62
Min (s)	1.0	1.0	1.0	1.0
25% (s)	132.0	77.0	48.0	34.0
50% (s)	235.0	151.0	106.0	117.0
75% (s)	635.0	425.0	207.0	291.0
Max (s)	35372.0	27121.0	29700.0	28952.0

For all workloads, these distributions characterize the workloads as 25% composed by long jobs, such that, long jobs in this context means jobs much longer than the majority of others. Looking for each workload, Table 7.1 shows that:

- 1w_03: 75% of the jobs are processed in less than 635s, and 25% are processed up to 35372s, which is 55 times the maximal processing time of short jobs.
- 1w_10: 75% of the jobs are processed in less than 425s, and 25% are processed up to 27121s, which is 63 times the maximal processing time of short jobs.
- 1w_17: 75% of the jobs are processed in less than 425s, and 25% are processed up to 29700s, which is 143 times the maximal processing time of short jobs.
- 1w_24: 75% of the jobs are processed in less than 291s, and 25% are processed up to 28952s, which is 99 times the maximal processing time of short jobs.

Because of this distribution, we decided to split, for each workload, the results of the simulations in two others, one composed of the jobs from the 75% of the distribution, and the other one composed of the jobs from the 25% of the distribution, respectively denoted by short_jobs and long_jobs. In the same idea will be denoted as all_jobs the original workload.

First, we will present in the next sections the analyses of results of all_jobs, then we will compare these with the analyses of short_ and long_ jobs aiming to point out some possible effect caused by the size of the jobs.

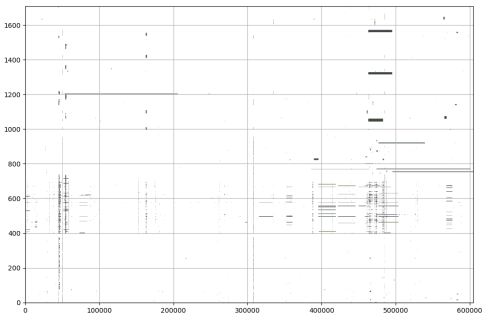
7.2 Job Allocation

We decided to illustrate the job allocation by Gantt Charts. These charts represent in x-axis the processing time of the jobs, which means, the time that a job spent to be executed. In the y-axis it shows the resources, which for us mean the QMobos. This way, it is possible to see rectangles representing the job's execution, such that, the size of the rectangles represent the size of the jobs, being visually distinguishable. In addition, by these charts it is possible to see the difference among the scheduling policies whenever we see the rectangles in different positions for each policy.

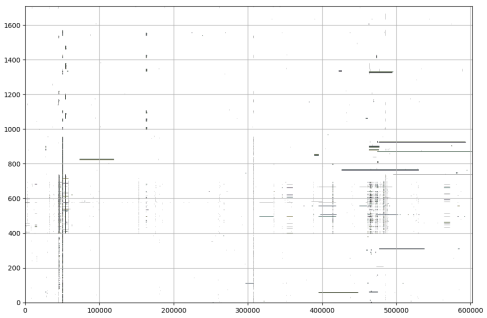
The Gantt Charts in Figure 7.1 represents the allocation of jobs for the workload of 1w_24 for the five presented policies. The sixth figure show all the previous five merged into one, super imposing it. Looking at these charts, it is possible to conclude that the allocation among the schedulers changes for each policy. Especially on the sub-figure *(f) All schedulers difference* because it shows the super imposing of all others charts, which means that, if the allocation would be the same, the super imposing would produce the same picture, which does not happen here.

7.3 Data Sets Dependencies

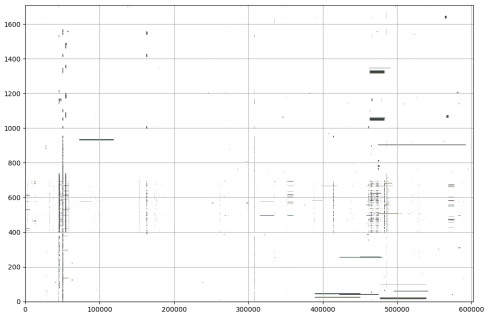
In order to investigate if the data sets affect somehow the scheduling policies, we will discuss in this section analyses regarding the data sets dependencies. We figured out that several QTasks depend on the same data sets, which could cause different results if considered at the scheduling decision phase, since Algorithm 1 does not consider this information.



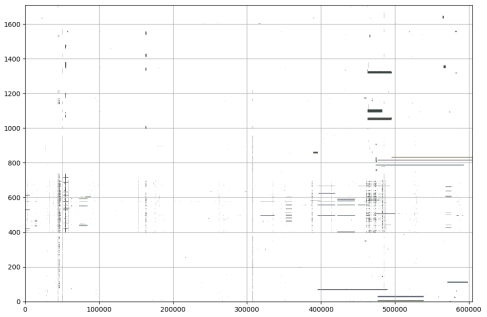
(a) Standard



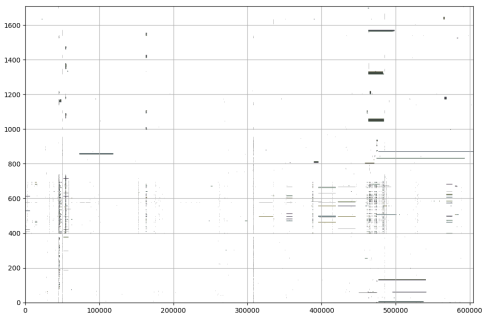
(b) Locality based



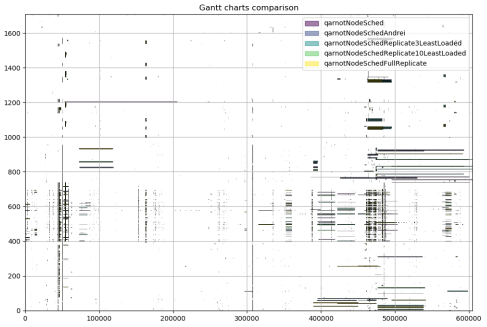
(c) Full replicate



(d) Replicate3



(e) Replicate10



(f) All schedulers difference

Figure 7.1 – Gantt Charts for the 1w_24 workload

The figures 7.2, 7.3, 7.4 and 7.5 show in x-axis the id of the data sets and in the y-axis the number of instances that depend on that data set. It is important to emphasize here two points, the first one is that a QTask is composed of many instances but, instances from the same QTask could be allocated on different QBoxes, if there are not enough available QMobos on the same QBox at the allocation phase. Then, these instances would require the data set transfer for two or more different QBoxes. The second one, as described in Section 5.3.2, a QTask depends on a list of data sets. Is important to emphasize here that, these figures will describe the number of instances requiring the same data sets, which does not mean that the sum of all bars totals the number of instances, because one could require the data sets with ID: 3, 5, 23 and 30, for example. In addition, the data sets with ID 1 represents *null* data sets, which means that the jobs are not dependent on any data set.

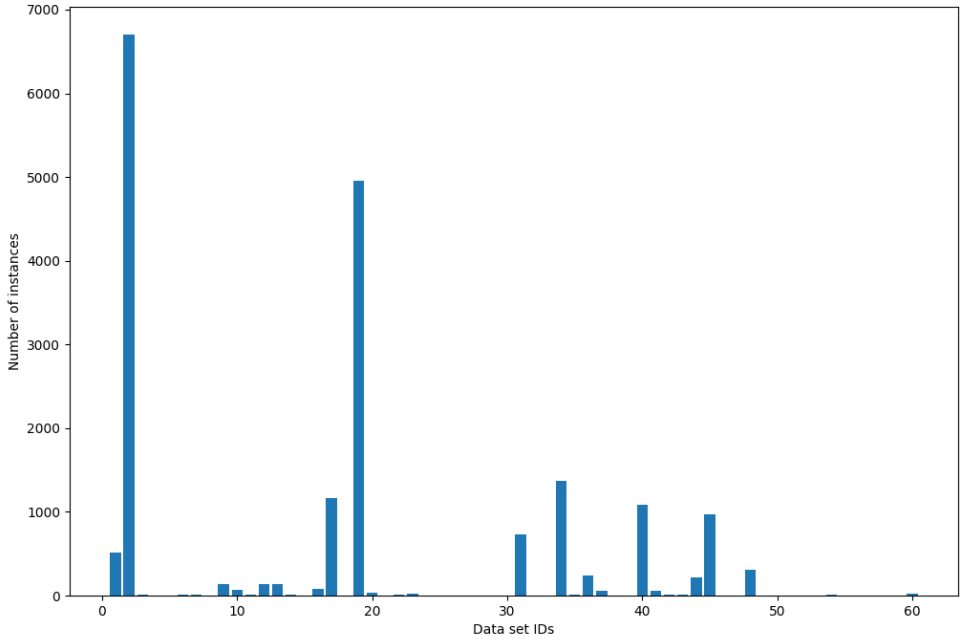


Figure 7.2 – Data sets dependencies for 1w_03 workload.
Number of instances: 7350. **Number of data sets:** 60

From Figure 7.2, one can see that the data set with ID 2 is required by about 6,700 instances, which represents 91% of the total number of instances. It is followed by the data set with ID 19, about 5,000 instances, representing 68% of the total number of instances. In the figure, one can also see other data set IDs reasonably required as 17, 34, 40 and 45, but, not so much as the two emphasized.

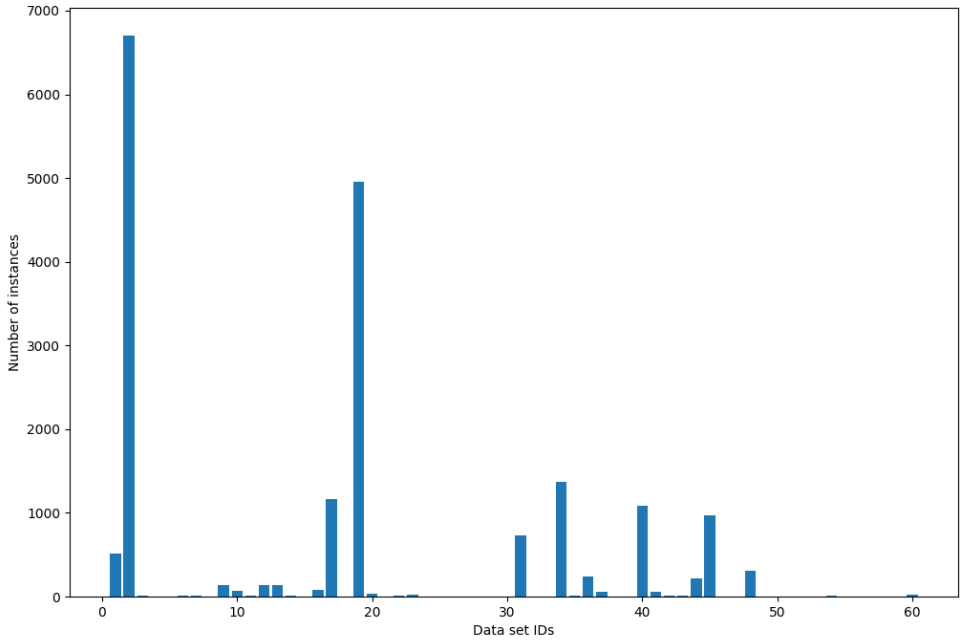


Figure 7.3 – Data sets dependencies for 1w_10 workload.
Number of instances: 5990. **Number of data sets:** 43

From Figure 7.3, one can see that the data set with ID 18 is required by about 2,900 instances, which represents 48% of the total number of instances. It is followed by the data sets with ID 34 and 16, about respectively 2,400 and 1,700 instances, representing 40% and 28% of the total number of instances. In the figure, one can also see other data set IDs reasonably required as 1, 15, 24, 33 and 37, but, not so much as the two emphasized.

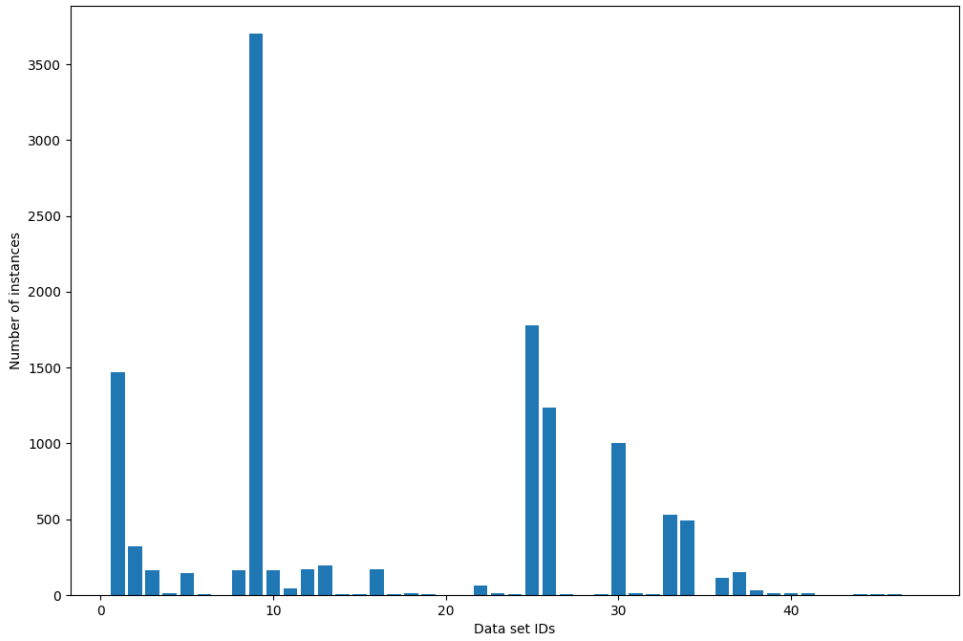


Figure 7.4 – Data sets dependencies for 1w_17 workload.
Number of instances: 5506. **Number of data sets:** 47

From Figure 7.4, one can see that the data set with ID 9 is required by about 3,700 instances, which represents 67% of the total number of instances. It is followed by the data sets with ID 25 about 1,800 instances, representing 33% of the total number of instances. In the figure, on can also see other data set IDs reasonably required as 1, 26 and 30, but, not so much as the two emphasized.

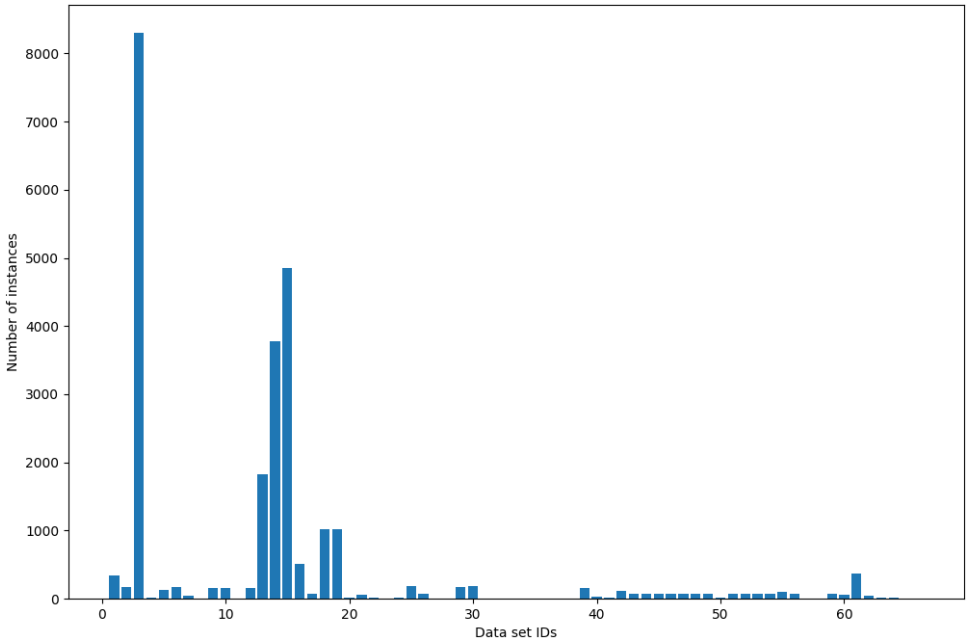


Figure 7.5 – Data sets dependencies for 1w_24 workload.
Number of instances: 8852. **Number of data sets:** 66

From Figure 7.5, one can see that the data set with ID 3 is required by about 8,500 instances, which represents 96% of the total number of instances. It is followed by the data sets with ID 15 about 5,000 instances, representing 56% of the total number of instances. In the figure, one can also see other data set IDs reasonably required as 13 and 14, but, not so much as two the emphasized.

Analyzing the data set dependencies we recognized that the four workloads extracted from Qarnot are not equally distributed in terms of data sets. In other words, it is possible to see for these ones that there are at least two very popular data sets among the instances. It could affect the scheduling policies and will be discussed in the next section.

7.4 Scheduling Metrics

In order to compare various scheduling strategies based on real-world traces of the *Qarnot* platform we will discuss in this section the number and the total size of data transfers, along with the bounded slowdown of the instances. The following discussion will be based on plots such that the x-axis represents the workloads, respectively 1w_03, 1w_10, 1w_17 and 1w_24. In addition, it is possible to see in x-axis the 5 scheduling policies implemented as described

in Chapter 6, respectively the FullReplicate, LocalationBased, Replicate10, Replicate3 and the Standard. The y-axis represents the metrics discussed.

7.4.1 Data Transfers

The number of transfers and the total size of data transfers are two important metrics if, for example, there are huge data sets which would take long time to be transferred, or in the case of limited resources in terms of storage disks, which will not support many data sets in the same machine.

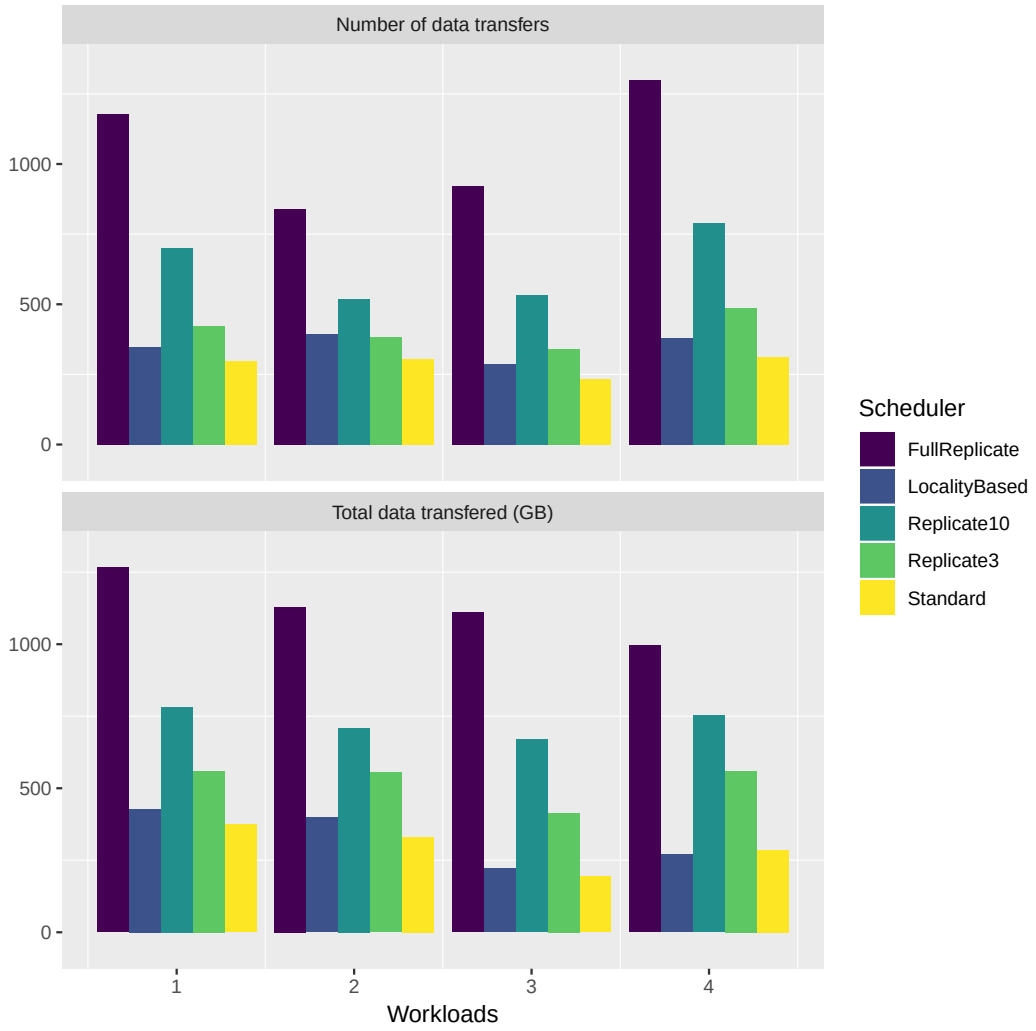


Figure 7.6 – Total data transferred for 1 week workload

From Figure 7.6 one can see that for both the *Number of data transfers* and the *Total size of data transferred (GB)*, the schedulers FullReplicate, Replicate10 and Replicate3 are the three with the highest values, with the exception of the Replicate3Least-Loaded for the workload 2 in the *Number of data transfers*. But, in general, this behavior is totally expected since, respectively, they replicate data sets in all, 10 and 3 QBoxes.

On the other hand, at the first time, we expected that the LocalityBased would reduce the number of data transfer compared to the Standard scheduler. But, as we analyzed the data sets dependencies, this behavior could be justified by the high popularity of few data sets among the workloads. We believe that what happens in this specific scenario is that the LocalityBased would reduce data transfers until a QBox get unavailable running as many instances as possible. Then, this very popular data sets should be transferred to other QBoxe and it would happen, maybe, during the whole simulation. This way, the LocalityBased got close or higher values between itself and the Standard scheduler, even the last one does not consider any location or data set information in the allocation decision phase.

We believe that if the data sets dependencies would be well distributed among the workloads, the LocalityBased policy would reduce the number of data transfers, but it is not our case and we would need to simulate this specific kind of data to validate it.

7.4.2 Bounded Slowdown

The bounded slowdown is a classical metric that has been utilized in the literature [17] and was implemented in the context of this thesis because it was not computed from the toolkit utilized. We computed the bounded slowdown as $bounded\_slowdown = max\{(waiting\_time + execution\_time)/max\{execution\_time, \tau\}, 1\}$, such that τ denotes a threshold, equals 1 for our experiments, since this is the minimum size of our instances as the table 7.1. Here we also considered that *waiting_time* of an instance depends on the time to transfer the required data sets and the time to decide in which QMobo the instance should be executed. This metric has been utilized to analyze if the waiting time of a job is proportional of its size. Considering that one important goal of scheduling policies is manage the waiting time of the jobs, it is an important metric. In addition, it is known that this metric is more sensitive for short jobs, since if the execution time is close to zero, this formula depends on the waiting time and the threshold. If the waiting time of short jobs is much bigger than its execution time, the bounded slowdown will be high, which does not mean that the execution time was big, but the inverse.

From Figure 7.7 one can see that the FullReplicate scheduler presents the lowest values for both measurements, the *Mean bounded slowdown* and *Max bounded slowdown*. For almost all the other cases, the Replicate10 and Replicate3 are the next lowest ones, with the exception of the *Max bounded slowdown* with the second and fourth workloads. But, in general, this behavior is also totally expected since these schedulers replicate much more data sets than the LocalityBased and Standard. Then, the waiting time of jobs managed by FullReplicate, Replicate10 and Replicate3 tends to be small, because it does not depend on the data transfer time, it just depends on the decisions process time, in general.

Following the same justification as Section 7.4.1, the LocalityBased presents close or higher values when compared with the Standard thanks the data sets dependencies. We believe that because of the waiting time of the data transfers, the instances managed by these schedulers wait more time than the others managed by the replicate based schedulers (FullReplicate, Replicate10, Replicate3). In addition, we believe that the LocalityBased presents, in general, a bounded slowdown bigger than the Standard because the LocalityBased does, also in general, more data transfers and transfers more data than the Standard.

To analyze in more detail, as we explained above, the bounded slowdown is more sensitive for short jobs than long jobs, then in the next section we will present comparisons looking for grouped instances by their sizes.

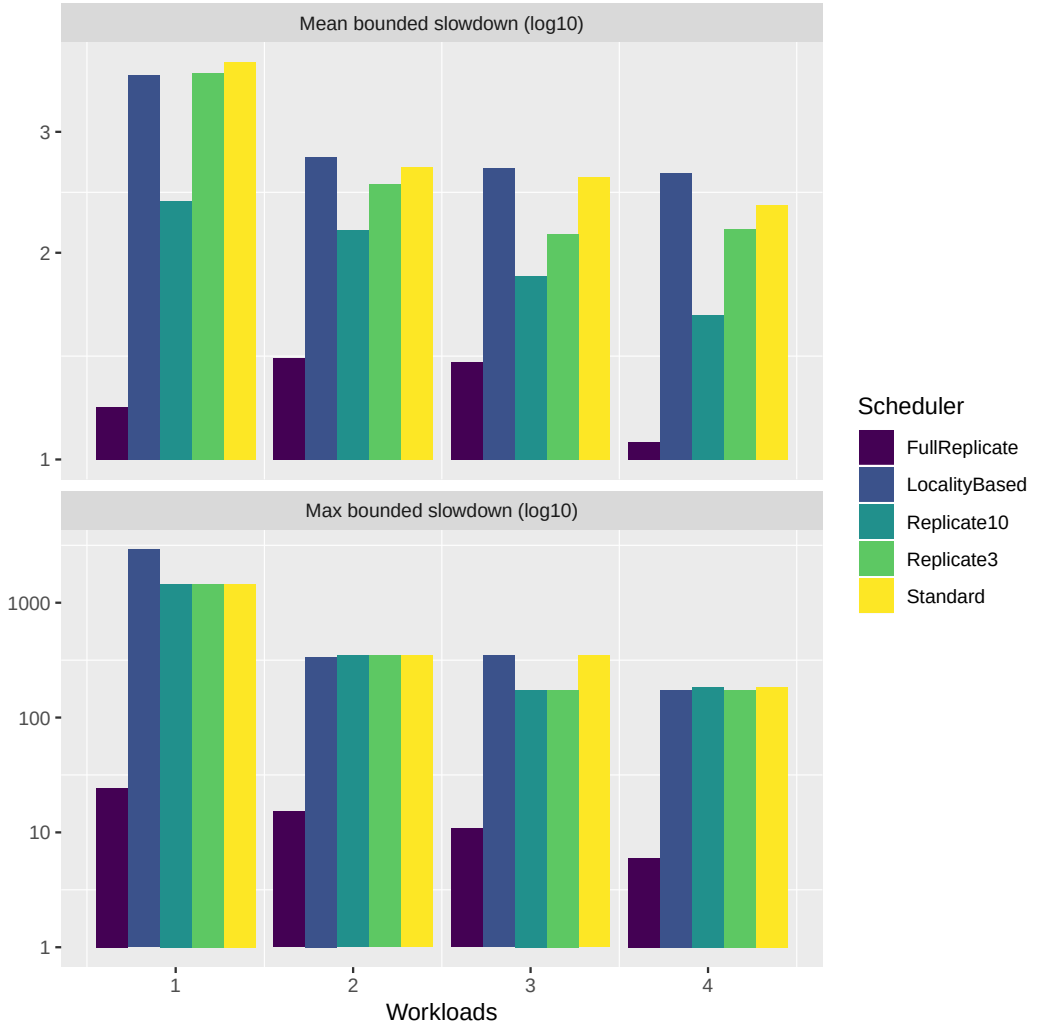


Figure 7.7 – Bounded Slowdown for 1 week workloads

7.4.3 Job's Size Effect in the Measured Metrics

Due to the characterization of 75% of the jobs as short and 25% as long, the results of the previous simulations were split into two others. This way we investigated the effect of the short and long jobs on the bounded slowdown. Once again, it is important to emphasize that the data come from the same simulation, we filtered the results from the *all_jobs* and split into two other, as *small_jobs* and *long_jobs*, as explained in section 7.1.

As one can see, the behavior in Figure 7.8 is, in general, the same as Figure 7.7. As this data is composed only for short jobs, its *execution_time* is low, then, we attributed the high values visible in Figure 7.8 to its *waiting_time*. Finally, as the replicate based schedulers (FullReplicate, Replicate10, Replicate3) present low values when compared with the others, we attributed that the *waiting_time* for the LocalityBased and Standard schedulers is impacted much more by the data transfers time than the allocation decision process time.

The behavior in Figure 7.9 is different when compared with Figure 7.7. Here the values for the FullReplicate are the highest ones and we justify it by the *waiting_time* from the allocation

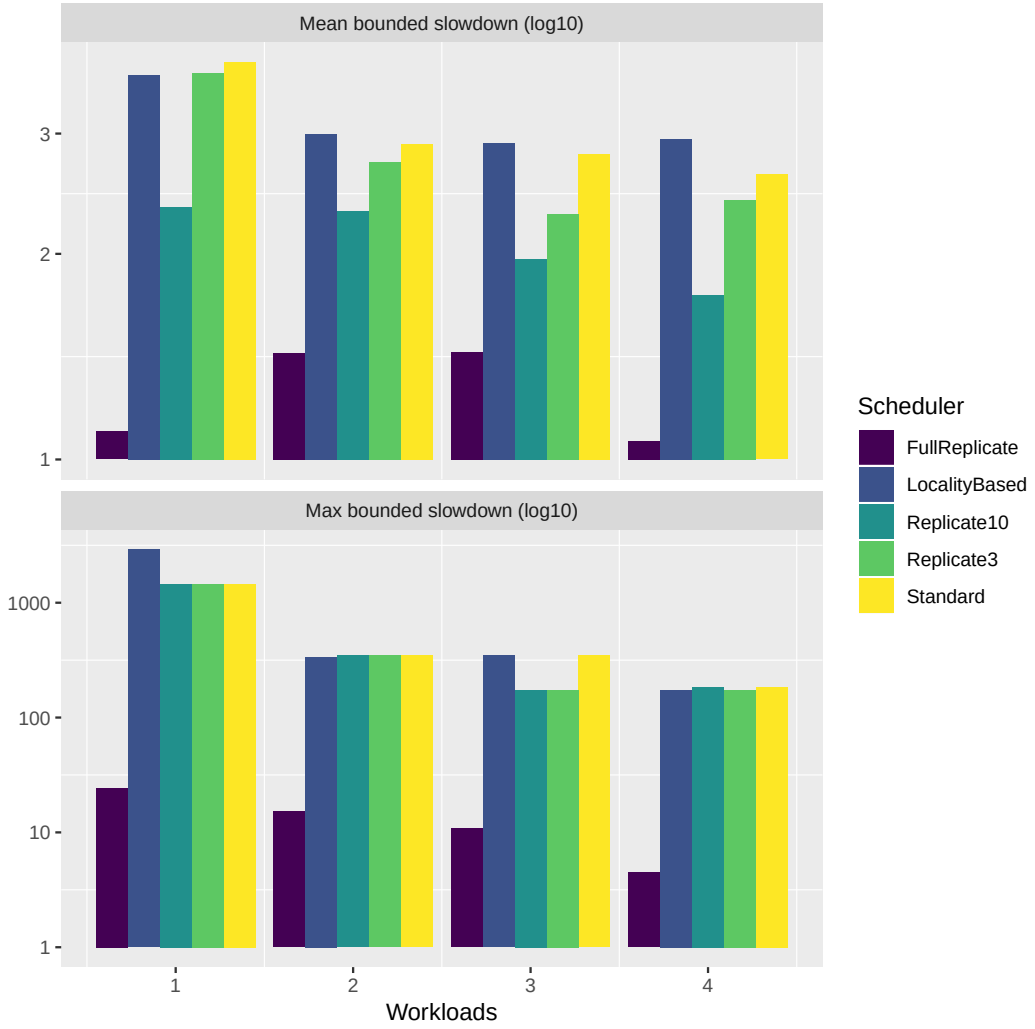


Figure 7.8 – Bounded Slowdowns for short jobs from 1 week workloads

decision process that, in general, takes more time than short jobs. But, is important to emphasize here that the values of Figure 7.9 in the *Mean bounded slowdown* are much lower than the presented in Figure 7.8, since that is not possible to see the second tick in the y-axis of this plot.

Comparing Figure 7.8 and Figure 7.9 one can see that the premise that short jobs are more sensitive to these metrics is true in our case, because the first figure presents much more high values than the second. And considering that 75% of the jobs are being represented in Figure 7.8 as the short jobs and 25% in Figure 7.9 as long jobs, we understood that the behavior of Figure 7.7 is much more impacted by the short jobs into these workloads.

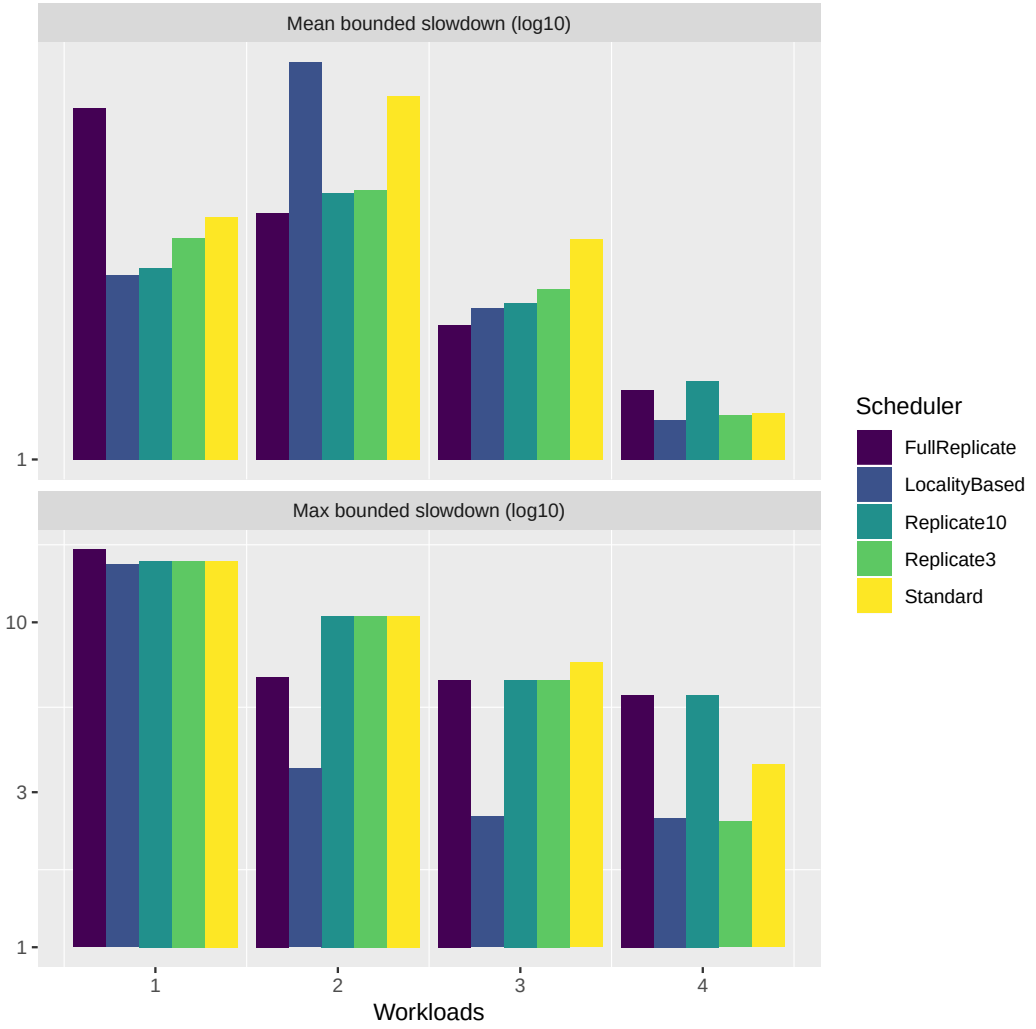


Figure 7.9 – Bounded Slowdowns for long jobs from 1 week workloads

7.5 Analyses of Results

All performed simulations were deterministic, then we ran one simulation with each scheduler for several inputs corresponding to 1 day, 3 days 1 week or 2 weeks of the *Qarnot* platform. The running time of one simulation was less than 5 minutes for a 1-day simulation, around 10 minutes for a 3-day simulation, less than 35 minutes for a 1-week simulation and around 50 minutes for a 2-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 number of transfers, the total transferred data and the mean and max bounded slowdown. In addition, we analyzed the job's processing time distribution and the data sets dependencies.

Due to lack of space, and as the simulations of the four periods (1-day, 3-days, 1-week and 2-week) lead to similar conclusions, we only present in Table 7.1 the results for the 1-week period.

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. Re-

garding the LocalityBased scheduler, the improvement of performance is quite low and even worse for all presented metrics, when compared with the Standard algorithm. This was quite surprising at first as the total transferred data actually increased, while the initial purpose of this scheduler was to leverage the data transfer already performed. But, regarding the data set dependencies it was justified. Such results, which are counter-intuitive, clearly illustrates the importance of validating the behaviors of scheduling algorithms through simulations before envisioning their real deployment.

The first thing to notice is that all variants improve the scheduling metrics compared to the standard algorithm of *Qarnot*, except for the mean bounded slowdown of LocalityBased, which is very close to the baseline, with a cost of an increasing size of the data transfers.

Comparing the replication policies, we can see that $Replicate3$ and $Replicate10$ 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 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 data sets 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 job scheduling policy for *Qarnot Computing* is still an ongoing action. And regarding the data sets dependencies, further strategies could predict the data transfers time and then, the scheduling policies could be mixed between a replicated one whenever a very popular data set is recognized, and locality based if the data sets would be eqaully distributed among the jobs.

— 8 —
Conclusion

8.1 Concluding Remarks

We utilized in this thesis a dedicated toolkit to evaluate scheduling 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.

This thesis presented how to use a simulated edge platform to easily evaluate existing placement strategies, even if the platforms complete validation was still an on-going work. 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.

We showed that to get a strategy as the best one, first of all, is important to know the workload that will be managed and its data-set dependencies. Thanks to our use case, we investigated several scheduling strategies and compared them to the actual policy implemented in the *Qarnot* platform. We showed that the workloads managed by the Qarnot are composed, in general, of 25% of long jobs and 75% of short jobs. We also showed that a majority of instances in these workloads require the same data sets. Finally, we showed that considering this context, the best strategies are those which take into account replication of data-sets.

We considered the work that has been developed in this thesis very important to the *Qarnot Computing* in the sense that it could be a very useful tool to simulate their strategies, intended modifications, to anticipate some behaviors and also to prepare environments and offices that would utilize their products. All of it thanks to a simulated tool, that allows engineers to perform this kind of studies without affecting their production system.

Since this context presents many challenges, we know that there are several possibilities to continue this work and the main future goal is to achieve the development of a Digital Twin, but still on going due to the difficulties presented in Chapter 2. But, we consider that this thesis was a step forward in this context.

8.2 Future Remarks in Edge and Cloud Computing

According to Shi et al. [38], [39] there will be 50 billion objects connected to the Internet by 2020, as predicted by Cisco Internet Business Solutions Group. Some IoT applications might require very short response time, some might involve private data, and some might produce a large quantity of data which could be a heavy load for networks. They conclude that

Cloud computing is not efficient enough to support these applications due to the growth of data production at the edge of the network. Therefore, it would be more efficient to also process the data at the edge of the network, close to where it is generated. And remark that previous work such as micro data center, cloudlet, and fog computing have been introduced to the community because cloud computing is not always efficient for data processing when the data is produced at the edge of the network.

Finally, several possibilities of next steps that could be taken in the future of Edge and Cloud Computing are discussed by Bittencourt et al [9]:

1. Fog and 5G for IoT: while the first 5G deployments are expected in the next couple of years, several challenges remain in how these deployments will support IoT services integrated with cloud and fog computing.

2. Serverless Computing: microservices management throughout the IoT-Fog-Cloud hierarchy presents challenges associated to the movement of services among IoT, fog, and cloud devices. The automatic adaptation of the execution of microservices must consider deployment location and context, but should also not neglect resource constraints that may exist at each level of the fog.

3. Resource Allocation and Optimization: The composition of devices in the IoT-Fog-Cloud continuum brings novelties as the heterogeneity of devices and applications reach unprecedented levels, then optimization in resource allocation becomes more challenging.

4. Energy Consumption: The proliferation of IoT devices and the ever increasing rate of data produced are increasing pressures on energy consumption. One should expect that such pressures will have to be addressed at both hardware and software levels as well as their interplay.

5. Data Management and Locality: There are several open issues related to data management and locality in IoT-Fog-Cloud computing systems. First and foremost, these systems are typically composed of a broad set of heterogeneous communication technologies such as cellular, wireless, wired, and radio frequency. This means that the systems orchestrator has to be able to handle distinct underlying networks as well as different addressing schemes.

6. Applying Federation Concepts to Fog Computing and IoT: Federations will be widely used in many different application domains. The outstanding challenge here is how can federation capabilities be best applied in fog and IoT environments? The easiest answer is to simplify the deployment and governance models to be used.

7. Trust Models to Support Federation in Fog and IoT Environments: Identity and trust are the cornerstones of federation management. While a number of methods exist for establishing identity and trust, the only feasible methods are based on cryptographic methods. An inherent property of IoT environments, though, is that the closer to the edge one gets, the more resource-constrained the devices will become.

8. Orchestration in Fog for IoT: Despite recent developments in the area of fog orchestration for the Internet of Things, there are still several open issues that need to be addressed. First and foremost, privacy must be tackled in accordance to the European Union General Data Protection Regulation as well as similar regulations being enforced worldwide.

9. Business and Service Models: While cloud computing has been offering a variety of business and service models through the years, it is not clear yet if fog computing can simply incorporate the cloud models or if new business or service models would be feasible.

10. Mobility: Efficiently allocating resources for mobile users is a challenge in fog computing. Users and devices mobility patterns are an important aspect to provide proper service when offloading to the fog occurs. Dealing with a large set of mobile users with diverse applications and requirements is a highly dynamic scenario, which makes resource management challenging.

11. Urban Computing: Although several research efforts related to urban computing have been performed recently, it is possible to find open issues and opportunities for studying cities and societies using location-based social networks (LBSN) data.

12. The Industrial Internet of Things: Designing software that exploits the Industrial Internet of Things constitutes a "system of systems" challenge. Taking into account the whole Iot-Fog-Cloud continuum, addressing the complexity of this challenge will require frameworks that enable interoperability but are also able to cope with varying and possibly conflicting user and system requirements.

Bibliography

[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] Kento Aida. Effect of job size characteristics on job scheduling performance. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–17, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[5] 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.

[6] 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.

[7] 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.

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

[9] Luiz Bittencourt, Roger Immich, Rizos Sakellariou, Nelson Fonseca, Edmundo Madeira, Marilia Curado, Leandro Villas, Luiz DaSilva, Craig Lee, and Omer Rana. The internet of things, fog and cloud continuum: Integration and challenges. *Internet of Things*, 3-4:134 – 155, 2018.

[10] 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.

[11] 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.

[12] 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.

[13] Anderson da Silva, Clement Mommessin, Pierre Neyron, Denis Trystram, adwait bauskar, Adrien Lebre, Alexandre Van Kempen, Yanik Ngoko, and yoann ricordel. Investigating Placement Challenges in Edge Infrastructures through a Common Simulator. In *Mascots 2019 - 27th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 1–16, Rennes, France, October 2019.

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

[15] 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.

[16] A. Essafi, D. Trystram, and Z. Zaidi. An efficient algorithm for scheduling jobs in volunteer computing platforms. In *2014 IEEE International Parallel Distributed Processing Symposium Workshops*, pages 68–76, May 2014.

[17] Dror G. Feitelson. Metrics for parallel job scheduling and their convergence. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 188–205, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[18] 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.

[19] F. C. Heinrich, T. Cornebize, A. Degomme, A. Legrand, A. Carpen-Amarie, S. Hunold, A. Orgerie, and M. Quinson. Predicting the energy-consumption of mpi applications at scale using only a single node. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 92–102, Sep. 2017.

[20] 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.

[21] Lu Huang, Hai-shan Chen, and Ting-ting Hu. Survey on resource allocation policy and job scheduling algorithms of cloud computing1. *Journal of Software*, 8, 02 2013.

[22] Hameed Hussain, Saif Ur Rehman Malik, Abdul Hameed, Samee Ullah Khan, Gage Bickler, Nasro Min-Allah, Muhammad Bilal Qureshi, Limin Zhang, Wang Yongji, Nasir Ghani, Joanna Kolodziej, Albert Y. Zomaya, Cheng-Zhong Xu, Pavan Balaji, Abhinav

Vishnu, Fredric Pinel, Johnatan E. Pecero, Dzmitry Kliazovich, Pascal Bouvry, Hongxiang Li, Lizhe Wang, Dan Chen, and Ammar Rayes. A survey on resource allocation in high performance distributed computing systems. *Parallel Computing*, 39(11):709 – 736, 2013.

[23] 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.

[24] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys Tutorials*, 19(4):2322–2358, Fourthquarter 2017.

[25] 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.

[26] Jie Meng, Samuel McCauley, Fulya Kaplan, Vitus J. Leung, and Ayse K. Coskun. Simulation and optimization of hpc job allocation for jointly reducing communication and cooling costs. *Sustainable Computing: Informatics and Systems*, 6:48 – 57, 2015. Special Issue on Selected Papers from 2013 International Green Computing Conference (IGCC).

[27] 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.

[28] 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.

[29] Y. Ngoko, N. Saintherant, C. Cerin, and D. Trystram. Invited paper: How future buildings could redefine distributed computing. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1232–1240, May 2018.

[30] C. Pahl and B. Lee. Containers and clusters for edge cloud architectures – a technology review. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 379–386, Aug 2015.

[31] S. M. Parikh. A survey on cloud computing resource allocation techniques. In *2013 Nirma University International Conference on Engineering (NUiCONE)*, pages 1–5, Nov 2013.

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

[33] Muhammad Bilal Qureshi, Maryam Mehri Dehnavi, Nasro Min-Allah, Muhammad Shuaib Qureshi, Hameed Hussain, Ilias Rentifis, Nikos Tziritas, Thanasis Loukopoulos, Samee U. Khan, Cheng-Zhong Xu, and Albert Y. Zomaya. Survey on grid resource allocation mechanisms. *Journal of Grid Computing*, 12(2):399–441, Jun 2014.

[34] M. Randles, D. Lamb, and A. Taleb-Bendiab. A comparative study into distributed load balancing algorithms for cloud computing. In *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, pages 551–556, April 2010.

[35] A. S. M. Rizvi, T. R. Toha, M. M. R. Lunar, M. A. Adnan, and A. B. M. A. A. Islam. Cooling energy integration in simgrid. In *2017 International Conference on Networking, Systems and Security (NSysS)*, pages 132–137, Jan 2017.

[36] M. Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, Jan 2017.

[37] 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.

[38] 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.

[39] W. Shi and S. Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, May 2016.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.