Distributed Grid computing Manager covering Waste Heat Reuse Constraints

Rémi BOUZEL
QLab
Qarnot computing
Montrouge, France
remi.bouzel@qarnot-computing.com

Yanik Ngoko
R&D
Qarnot computing
Montrouge, France
yanik.ngoko@qarnot-computing.com

Paul Benoit
CEO
Qarnot computing
Montrouge, France
paul.benoit@qarnot-computing.com

Nicolas Sainthérant
Innovation
Qarnot computing
Montrouge, France
nicolas.saintherant@qarnot-computing.com

***Abstract*—In this paper, we discuss a green and distributed type of datacenters, implemented by Qarnot computing. This approach promotes a new computing paradigm in which computers are considered as machines that produce both computation and heat, and are therefore able to reuse the waste heat generated. It is based on two main technologies: a new model of servers and a new distributed grid computing manager which encloses a heat aware job scheduler. This paper focuses on the infrastructures and cloud computing services that were developed to answer the constraints of this new HPC paradigm. The description covers the job scheduler that ensures security and resilience of Qarnot distributed computing resources in a non-regulated environment. We summarize the key computational challenges met and the strategies developed to solve them. A specific use case is detailed to show that, in spite of its thermal-aware specificity, spawning a job on the Qarnot platform remains as simple as on any other state-of-the-art job scheduler.**

***Keywords*—Green HPC, Job scheduler, Edge Computing, Heating as a service, Resource manager**

I. Introduction

In the pure-compute paradigm, the heat produced by computing machines is an unexpected outcome. The execution of a computer program must pursue the sole objective to produce the correct output data. On modern computing machines, the pure-compute paradigm led to the usage of fans in personal computers or cooling technologies like chilled water and air conditioning systems in datacenters.

The pure-compute paradigm follows the vision that pioneers like Babbage or Alan Turing had about the functioning of computers. It is also supported by the impact of high temperatures on processors aging and reliability [1]. It led to centralized infrastructure known as datacenters which raise challenges in terms of cost and energy efficiency.

Alternatively to pure-compute, there is the compute-and-heat paradigm. Here, the computer programs are not only supposed to produce data. The heat is an expected outcome and computer designers should invest in the mastering of its generation. Though heat generation was already observed in successive generations of electronic computers, it is only in recent years that this second vision was clearly set with the development of heat recovery systems in datacenters [2] and in the pioneer initiative of Qarnot computing.

The remainder of this paper is organized as follows. In Section 2, we present and discuss the related works. Section 3 describes the specific infrastructures developed by Qarnot computing and the Q.Ware: the job scheduler that spawns jobs on the infrastructure of the distributed datacenter. A zoom on the specific constraints and solutions of this model is proposed in Section 4. Finally, Section 5 focuses on a use case example running an OpenFOAM job, before concluding in Section 6.

II. Related Work

Several resource managers were proposed for geo-distributed clouds.

In [3], the authors showed how to combine a classical grid scheduler with a set of clouds that offers infrastructure as services. In their model, the user requests describe a job to process. The jobs are submitted to the grid scheduler with an additional description of the machines to use for processing purposes. Depending on the availability of the different cloud providers it is connected to and the computing requirements, it will then boot the virtual machines on the most appropriate could providers. The grid scheduler then deploys the job on the virtual machines that are started.

OpenStack developed several solutions for the creation of distributed clouds. This includes distributed scheduling mechanisms implemented in Nova or the OpenStack cascading solution that can be used for a hierarchical management of several OpenStack sites.

The Nebula Edge Cloud [4] proposes a distributed edge computing cloud. It enables distributed data-intensive computing through a number of optimizations including location-aware data and computation placement, replication, and recovery.

The SlapOs cloud [5] introduced a distributed resource manager for cloud that implements the master/slave paradigm. The master nodes are stateful while slaves nodes are stateless. Slaves request to their master node which software they should deploy and which tasks to run. They frequently send reports to the master about their state.

Cooling-aware and thermal-aware placement was considered in [6]: the job scheduler is associated with a job placement algorithm. Servers are ranked based on their thermostat's demand: the higher the temperature required, the better rank; the job placement algorithm then spawns the incoming jobs on the servers, starting with the best ranked nodes, which allows to globally increase the temperature of the thermostat and reduce the energy consumption.

Finally, the distributed cloud management has also been envisionned throughout multi-cloud brokers that interact with several clouds to find the best resources according to user service level agreement [7].

Several other resource managers were proposed for geo-distributed clouds. For an extended state-of-the-art, we invite the interested reader to the report provided in [8]. Despite the existence of such solutions, they do not all implement one of the main features required in the Qarnot vision, which is the need to define a thermal-aware manager.

III. Qarnot Solutions: green Infrastructures and Job Scheduler

Qarnot distributed datacenters rely on two main technologies (See Fig. 1):

- Servers that can both run heavy computations and optimize the reuse of the heat through heated air (Q.Heaters or QH1, Q.Racks or QS1) or heated water (Q.Boilers or QB1). These servers, also referred to as Q.Rads, are installed as close as possible to the heat demand: in housings, buildings, and warehouses,
- A job scheduler (Q.Ware) that spawns jobs in order to fit the computation demand and the heating needs.

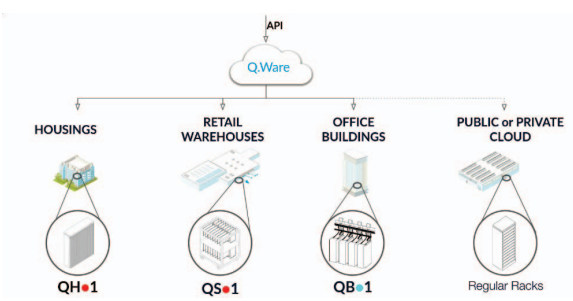

Fig.1. Qarnot Solutions

Qarnot computing promotes a utility computing model in which computing and heating are delivered from a single cloud infrastructure. The model is implemented by means of a geo-distributed cloud platform based on special server nodes of three different types: the heater, the rack, and the boiler. The goal is either to diffuse or collect the waste heat produced by computing to ensure proper IT operation.

The computing heaters or QH1 (See Fig. 2) embeds several processors connected to a large aluminium heatsink for heat diffusion in its environment. Q.Heaters are deployed in homes, offices, schools, etc.

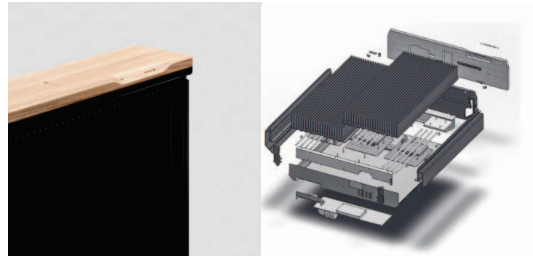

Fig.2. Qarnot digital Heater (QH1)

The Q.Racks follow the same principle but embed 16 servers on a rack. We use free cooling to both reduce the temperature of the computers and heat the room. They are essentially deployed in warehouses.

The Q.Boilers (See Fig. 3) embed 24 servers connected to aluminum "cold plates" used for heat dissipation and recovery. Cold inlet water flow is circulating through these plates in copper tubes and receives up to 3kW of waste heat. This water can finally be used as domestic hot water as part of a complete hydraulic system using hot water storage and production complement.

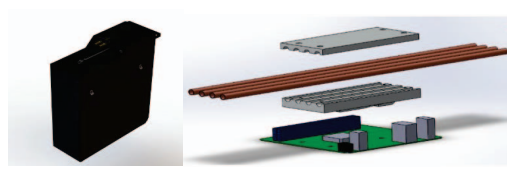

Fig.3. Qarnot digital Boiler (QB1) and 3D Model of its Dissipation aluminium Block

In all Qarnot products, CPUs are cooled with passive cooling: the heat is transferred from the CPUs to the air or the water. If needed, we voluntarily regulate the CPU temperature (for instance by reducing its frequency) in order to maintain the CPU in good condition.

The network of Q.Rads is the physical infrastructure of the distributed datacenter. In order to access such infrastructure, one must use the Q.Ware grid manager: a Qarnot computing orchestrator that handles the resource management and the specific constraints raised by the compute-and-heat paradigm.

The design of the Q.Ware system is based on several guiding principles, including the following ones:

- **Distributivity**: Q.Ware assumes a physical geo-distributed topology for edge computing. As for Qarnot cloud instances, the compute nodes are composed of Q.Heaters; Q.Racks, and Q.Boilers deployed in apartments, offices, and warehouses.
- **Security**: Q.Ware integrates state-of-the-art security practices:
  - bare-metal nodes: only one user on each physical server,
  - diskless nodes,

- usage of TPM modules,
- possibility to split the computation in chunks spawned in different locations,
- regular security audits.

- **Thermal-awareness**: As already mentioned, in addition to traditional cloud computing requests, Q.Ware also handles heating requests.
- **Multi-clouds** and multi-architectures: Q.Ware assumes a generic view of computing resources. They can consist of a motherboard (part or not of a Q.Rad), a connected device, a computer server. Q.Ware also supports various types of virtual machines and containers.
- **Scalability**: The Q.Ware is based on a hierarchy of 3 levels of servers: Q.Node, Q.Box, Q.Rad (Q.Heater, Q.Boiler, or Q.Rack) (See Fig.4). Thus, the Q.Rads support the clouds' abstractions at home level, the Q.Boxes aim at managing abstractions at building levels and Q.Nodes, abstractions at national level. We can easily add new compute nodes to Q.Ware by creating new instances of the different servers. The architecture is described more in depth in Section 4.

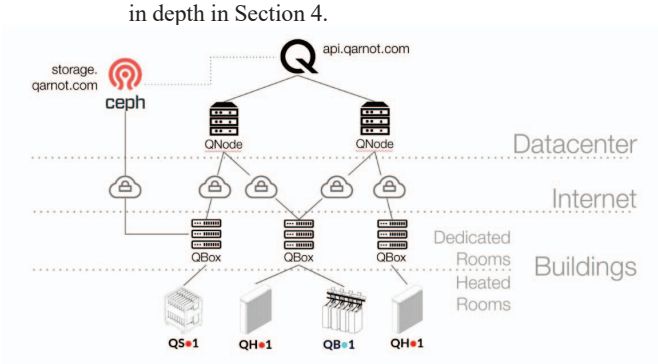

Fig.4. Qarnot Architecture Overview

- **Autonomy**: In Q.Ware, if a top server fails, the servers underneath will autonomously take the appropriate decisions to ensure that the heating is serviced. In addition, if there are no computing requests, the servers will automatically request computations from databases of scientific problems like Folding@Home [9].
- **Fault-tolerance and resilience**: A Q.Box can be connected to several Q.Nodes. This is important for continuing the service provisioning when a Q.Node fails. In the same way if a Q.Box could not be accessed, the Q.Rads will ensure the heating supply. This is described in details in Section 4.

Finally, the Q.Ware job scheduler includes a C# API and several SDK (Python, NodeJs, C#, CLI) for the submission of computing requests.

IV. Specific Constraints of a Green Distributed Datacenter

A. Intermittence of the Resources

Our infrastructures' intermittence is inherent to the distributed datacenter model as redundancy is much more complex to achieve in such an architecture, and a host might decide to switch off its heater at any moment. There are four main sources of intermittence for this architecture:

- internet outages (low redundancy),
- electricity breakdowns (no redundancy and no UPS),
- daily changes in the heat demand of the hosts (decreasing temperature or switching off a heater),
- seasonal changes in the heat demand of the hosts (less heat demand in summer).

Internet and energy redundancy is difficult to achieve because of the high number of sites hosting infrastructures, and the fluctuation of the heat demand is a necessary constraint to integrate as we aim to recycle the processors waste heat.

This intermittence is our main difference with large datacenters and its causes are also what allows this distributed datacenter to reuse the waste heat while keeping a high quality heating service. Properly mitigating these risks is a key challenge from the compute viewpoint as it has a great impact on the Service Level Agreements with computing clients.

In the following subsection, we explore the mathematical translation of changes in the heat demand through the opposite viewpoints of the client (compute) and the host (heat).

B. Client (compute) Viewpoint and Host (heat) Viewpoint

Scheduling in Q.Ware is naturally a multi-objective problem. This is because there are at least two viewpoints: the one of customers that want to compute (HPC customers) and the viewpoint of customers that want to be heated (host).

In Q.Ware, the **viewpoint of the HPC customers** is what we find in classical distributed scheduling systems: the goal is to get the results of submitted jobs as soon as possible. For this purpose, the current Q.Ware implementation focuses on makespan C_{max} minimization (time that elapses from the start of the job to the end).

The **viewpoint of the hosts** completely differs from what we could find in classical scheduling theory. Indeed, let us assume that at date t, a host of the heater wants to be heated at the temperature $target_i(t)$. Let us also assume that $ambiant_i(t)$ is the current ambient temperature observed from the heater. Given n jobs to schedule on m heaters, the hosts expect that the processing of the jobs should be done such as to minimize the difference between the target and ambient temperatures. This is captured with the objective function (1).

$$minimize \max_{1 \leqslant i \leqslant m} \int_0^{T_m} \left| target_i(t) - ambiant_i(t) \right| . dt \qquad (1)$$

First, these two viewpoints must match at a global scale. To do so, two global mismatch situations must be avoided:

- not enough computations submitted to match the heat demand: as exposed in Section 3, if there are not enough computing requests, the servers automatically request computations from databases of scientific problems,
- not enough heat demand to match the computation needs: to avoid this situation, it is necessary to always make sure that there are more servers deployed than necessary.

Once these two policies are implemented, the problem can be addressed at a more local scale. A general approach for solving multi-objective problems consists of formulating the other objectives as a constraint such as to have a single objective problem [10]. This is done in Q.Ware where the hosts viewpoints are defined as constraints in an adaptive scheduling approach.

C. *Intermittence Mitigations*

All jobs don't have the same sensitivity to real time and therefore to intermittence of resources. Q.Ware is especially used for batch computing and massively parallel computations, which are less impacted by the intermittence. In parallel, several solutions exist to mitigate the intermittence risks raised above:

Internet breakdowns:

- our biggest installations are equipped with redundant internet,

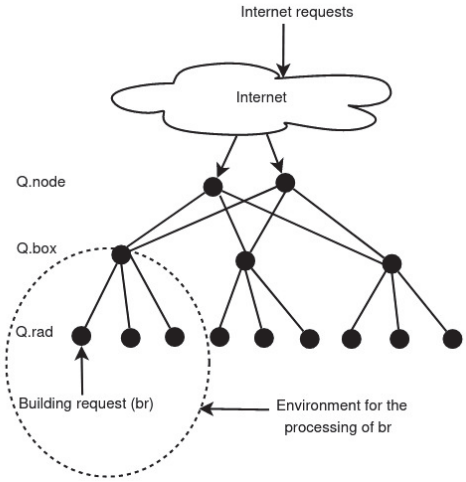

Fig.5. Difference between Q.Ware regular usage (internet request) and GRECO local usage (building request)

- it is possible to spawn a job on a local network if its building is equipped with Q.Heaters or Q.Boilers, without connecting to the Internet (See Fig. 5), see ANR GRECO project [11]. Besides bypassing the internet network, GRECO brings other interesting features to our platform: increase of the security for companies concerned with the transit of their data through the internet, increase of the bandwidth for data transfer, decrease of the ecological impact of the internet network.

Electricity breakdowns:

- when it happens, the job is spawned on another computing node, it is therefore seamless for the user as long as real time results is not a constraint,
- we work on local smart grid projects that would ensure computing nodes alimentation during breakdowns: the CATALYST H2020 project [12] aims at leveraging datacenters flexibility for the benefit of smart energy grids, including virtual machine live relocation across datacenters. This shall contribute to IT grid resiliency in accordance with local / regional energy (electricity and heat) constraints.

Daily changes regarding heat demand:

- if a server reached its heat request while a computation is ongoing, the job will be interrupted and spawned on another available node (when possible using checkpoints to not start the computation from scratch). However, as on any heater, the thermal inertia imposes a slow reactivity to the demanded change of temperature of the room, the job migration can therefore be taken care of in better conditions,
- diversity of our infrastructures: the business model of Q.Boilers, and of boilers in general, brings more flexibility as the objective is to always deliver the same power to the water, there is therefore no fluctuation in the demand and no job migration necessary,
- diversity of our locations: an important part of our computing infrastructures are installed in warehouses that need to be heated (between 10 000 and 20 000 physical cores per warehouse), which brings much more flexibility in heat management: spawning 50 more jobs on such infrastructure has a minor impact on the target temperature of the warehouse considering the number of cubic meters to heat and therefore brings more flexibility.

Seasonal changes regarding heat demand:

- the Q.Boilers bring more flexibility to our distributed datacenter as, during the installation, it is dimensioned to only serve the percentage of hot water that is used continuously over the year: shower, washing machine, dishwasher, etc, the remaining demand (for instance winter peak) being taken care of by regular boilers installed within the same hydraulic installation. The Q.Boilers are therefore used with the same intensity during the whole year to cover heat demand baseload,
- in the warehouses, it is possible to extract the heat without affecting the people working there (the servers are usually located in a closed room) and with a better efficiency than A/C which is used in classical datacenters. Such an efficiency can be achieved thanks to the lower density of servers in our warehouses compared to regular datacenters.

In the following subsection, we focus on the architecture and the way it handles broken connections that can result

from internet breakdown, electricity breakdowns, and changes in the heat demand.

D. An Architecture and a Grid Manager designed to handle Intermittence

The Q.Ware is composed of three types of servers: Q.Nodes, Q.Boxes and Q.Rads (Q.Rads include Q.Heaters, Q.Boilers, and Q.Racks). The servers are organized in a hierarchical tree where the Q.Nodes are root nodes and the Q.Rads are leaf nodes. An example of deployment is given in Fig.4. Here, each home is associated with a set of Q.Rads that are controlled at the building level by a Q.Box. The Q.Boxes are controlled by a Q.Node. In practice however, this deployment is not suitable. For fault tolerance issues, it is more interesting to link a Q.Box to several Q.Nodes: each Q.Box receives payloads from several Q.Nodes and runs the payloads in the order they were received. Ideally, Q.Ware assumes that the computing power of the cloud is in the processors inside the Q.Rads, but, as already mentioned, it can also manage these resources without the digital heater abstraction. In particular, Q.Ware can exploit concurrently the compute nodes of a Q.Rad and container/virtual machines deployed in another datacenter.

In the server hierarchy of Q.Ware, the heating requests are first routed to the leaf nodes while the computing requests are routed to the top. For computing requests, Q.Ware supports an API that is used to formulate the computing requests as the submission of a set of tasks. Here, a task refers to the execution of a container/virtual machine image and is associated with a set of input data and an output directory.

Each Q.Node runs a scheduler that manages a queue of tasks. The scheduler implements a list scheduling algorithm with priority on tasks. The principle of the algorithm is to iteratively loop on the queue and to select the tasks of higher priorities to deploy. Once done, several filters are applied to determine the compute nodes that will run the tasks. It is important to retain that the vision that the Q.Ware has of the compute nodes comes from data reported by the Q.Boxes. Thus, when for instance, in a heater, a request is sent to not produce any heat, the Q.Box will state that the corresponding nodes are not exploitable. In the list scheduling algorithm, there are globally three classes of priorities: background, high and low. Tasks of background priorities correspond to those that are mainly deployed to heat. Those with high priorities are associated with a computing request that has a strong SLA. The list scheduler of the Q.Node assumes that tasks are preemptable. However, we cannot interrupt a task with a given priority to run another of lower priority. At the lower end of the Q.Ware architecture, there are the Q.Rads servers that have direct access to the processors, sensors such as temperature and the main computing power information. The servers are also connected to a control interface (HMI) that the hosts of the heater can use to control their temperature.

The Q.Rad is able to negotiate computing loads with Q.Nodes, through the secured connection of its Q.Box. In the case the communication is broken between the Q.Box and the Q.Nodes (see the intermittence causes discussed above), the Q.Rads are still able to compute cached tasks. In the case where the communication with the Q.Box is broken, the Q.Rads will autonomously ensure that the heating will be serviced in launching a generic benchmarked computer program.

Over the Q.Rads and under the Q.Node, Q.Boxes are acting as local controllers to handle heating and jobs' dispatching, security and caching. The Q.Boxes manage the storage used in the Q.Rads: the disks are located in the Q.Boxes and the data is made available to the servers through NFS protocol. The choice to separate the management of the storage was initially driven by security and resilience considerations (easier to manage a disk outage in a Q.Box than in a Q.Rad). Q.Boxes are in charge of dynamic container deployment, input, output and session data synchronization with parent Q.Nodes. A Q.Box can also stop or pause a container of one of the Q.Rads it controls. Those decisions are based on collected sensors data. Finally, each Q.Box is connected to several Q.Nodes, so that if one of them failed, the Q.Box can still receive jobs to spawn.

E. Specificities of the Qarnot cloud

From the heat viewpoint, the Qarnot products deliver the same service as regular heaters and boilers. From the computing viewpoint, the distribution of the grid, which is necessary to reuse the servers' heat, has an impact on the complexity of the orchestrator:

- the Qarnot infrastructure is especially suited for distributed computations (e.g. Monte Carlo computations, parametric computation, etc.). For fine-grained parallel computations, all the nodes of the cluster must run in the same location (for instance in a Q.Rack of a warehouse) in order to ensure a low latency in the nodes' communication. This adds an extra stress on the orchestrator, forcing Qarnot to pay attention to the diversification of the computations: the more diversity (distribution, parallelism), the better; while most datacenters don't need to make a specific effort on this aspect,
- because of installations in offices and apartments without energy redundancy, Qarnot can't offer tier IV SLAs (>99,995% availability) for all its sites. However, thanks to the higher energy control in the industrial warehouses installations, it is possible to redirect computing users with higher SLA needs towards these buildings. It is therefore possible to accomodate all computing users, but at the cost of adding more stress on the orchestrator and the installation strategy, that need to make sure we have enough installations with high standards SLAs to answer the demand.

V. Two use Cases

Even though Q.Ware includes more constraints than regular orchestrators, its API and SDKs were developed to seamlessly spawn a job in a few lines of code. Once the script is run, a Docker container will be deployed in one (or several) computing node(s) and the computation will start. As a hello world, Fig. 6 fshows a python script that simply prints "hello world from node ${INSTANCE_ID}!" where $ {INSTANCE_ID} will take the id of the program instance (here 0, 1, 2, 3):

```
#!/usr/bin/env python

import qarnot

# Create a connection, from which all other objects will be
# derived, using your secret token
conn = qarnot.connection.Connection(client_token="mytoken")

# Create a task, give it a name and choose the nb of
# instances of the program
task = conn.create_task('helloworld', 'docker-batch', 4)

# Set the command to run when launching the container
task.constants['DOCKER_CMD'] = 'echo hello world from node
                                ${INSTANCE_ID}!'

# Submit the task to the Api
task.submit()
```

Fig.6. Script spawning a hello world job

Fig. 7 shows how to spawn a real life job, in this case solving a multiphysics problem with OpenFOAM. To achieve this, we need to include storage buckets for the input (that will include all the input files necessary to the computation) and output data, and to select the proper Docker image that fits the needs of the computation (here one containing OpenFOAM and its dependencies), either a home made image or an existing one from the Docker hub (or another registry):

```
#!/usr/bin/env python
import qarnot

# Create a connection, from which all other objects will be
# derived, using your secret token
conn = qarnot.connection.Connection(client_token="mytoken")

# Create a task, give it a name and choose the nb of
# instances of the program
task = conn.create_task('openfoam_test', 'docker-batch', 1)

# Create a resource bucket and add an input file
input_bucket = conn.create_bucket('input-resource')
input_bucket.add_directory('path')

# Attach the bucket to the task
task.resources = [ input_bucket ]

# Create a result bucket and attach it to the task
output_bucket = conn.create_bucket('output')
task.results = output_bucket

# Choose the docker image and tag
task.constants["DOCKER_REPO"] = "qarnotlab/openfoam"
task.constants["DOCKER_TAG"] = "v1912"

# Set the command to run when launching the container
task.constants['DOCKER_CMD'] = "/job/Allrun"

# Submit the task to the Api
task.submit()
```

Fig.7. Script spawning a multiphysics job

VI. Conclusion

In this paper, we introduced Q.Ware, a new resource manager for a utility computing approach in which heating is considered as a valuable output. We described the specificity of the infrastructures, how some of them are structurally intermittent, and the details of the grid manager. We then focused on the solutions developed to incorporate seamlessly these new constraints without decreasing the quality of the scheduling or increasing its API complexity. We underlined the latest point by providing two short scripts that spawn jobs on this grid manager.

Q.Ware could pave the way for a new vision of distributed computing where cloud-services and heating are provisioned from a unique platform. This invites (1) to reconsider the question of cooling in datacenters and servers and (2) to design new scheduling algorithms for the efficient production of heat based on computers.

References

[1] Huang, L., Xu, Q.: Agesim: a simulation framework for evaluating the lifetime reliability of processor-based socs. In: Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2010, pp. 51–56. European Design and Automation Association, Belgium (2010)

[2] Alfonso, C., Giulio, P.: Cooling systems in datacenters: state of art and emerging technologies. In: Sustainability in Energy and Buildings: Proceedings of the 7th International Conference, SEB 2015, pp. 484–493. Elsevier (2015)

[3] Armstrong, P., et al.: Cloud scheduler: a resource manager for distributed compute clouds. CoRR abs/1007.0050 (2010)

[4] Ryden, M., Oh, K., Chandra, A., Weissman, J.: Nebula: Distributed edge cloud for data intensive computing. In: 2014 IEEE International Conference on Cloud Engineering (IC2E), pp. 57–66, March 2014

[5] Smets-Solanes, J.P., C´erin, C., Courteaud, R.: Slapos: a multi-purpose distributed cloud operating system based on an erp billing model. In: 2011 IEEE International Conference on Services Computing (SCC), pp. 765–766, July 2011

[6] E. K. Lee, H. Viswanathan and D. Pompili, "Proactive thermal-aware resource management in virtualized HPC cloud datacenters", IEEE Trans. Cloud Comput., vol. 5, no. 2, pp. 34-248, Apr.-Jun. 2017.

[7] Buyya, R., Barreto, D.: Multi-cloud resource provisioning with aneka: a unified and integrated utilization of microsoft azure and amazon ec2 instances. In: International Conference on Computing and Network Communications, pp. 222–235, December 2015

[8] Jennings, B., Stadler, R.: Resource management in clouds: survey and research challenges. J. Netw. Syst. Manage. 23(3), 567–619 (2015)

[9] The Folding@home project uses all the unused computing resources that are made available to it (by installing its software on a laptop, desktop, server, etc.) to run molecular dynamics simulation that help disease research to move forward.

[10] Dutot, P.F., Rzadca, K., Saule, E., Trystram, D.: Multi-objective Scheduling, Chap. 9. Chapman and Hall/CRC Press, November 2009. ISBN: 978-1420072730

[11] The GRECO project: See https://anr.fr/Projet-ANR-16-CE25-0016 for more information.

[12] The CATALYST project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 768739. See https://project-catalyst.eu/ for more information.