Heating as a Cloud-Service, A Position Paper (Industrial Presentation)

Yanik Ngoko(✉)

Qarnot Computing, 92120 Montrouge, France
yanik.ngoko@qarnot-computing.com

Abstract. In this paper, we discuss a novel utility computing approach, implemented by the company Qarnot computing in private clouds. The approach promotes a new computing paradigm in which computers are considered as machines that produce both data and heat. It is based on two main technologies: a new model of servers and a new resource manager for servicing both computing and heating as a cloud-service. This paper focuses on the resource manager promoted by this utility computing approach. We summarize the architecture of the middleware and describe the key computational challenges. We also provide a performance characterization on the thermal comfort and processing time. Some preliminary results show that the proposed utility computing approach can lead to distributed systems that are competitive with both traditional cloud solutions and heating systems.

Keywords: Heating as a service · Utility computing · Resource manager

1 Introduction

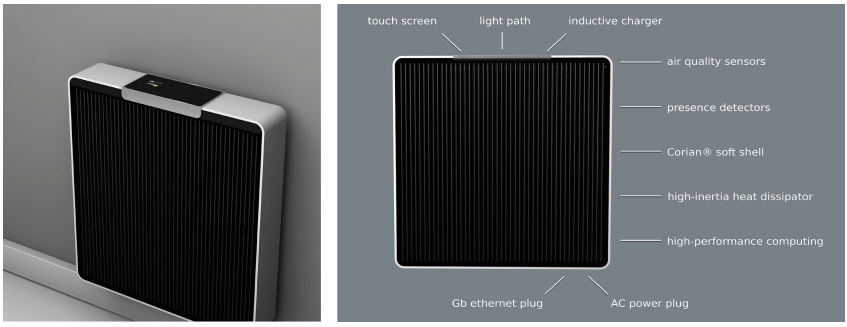
With the analytical engine [7], Charles Babbage conceptualized the main ideas of a general purpose *computer*. One of the most innovative point of the Babbage machine was the input/output model he defined. This model is still the reference today; indeed, it is commonly admitted that the computers are machines that given an *input data* and a *computer program* will automatically generate *output data* that are the results of the computer program on the input. The Babbage analytical engine was a mechanical machine. Many years after its conceptualization, the electronic design of computers stressed the importance of the heat generated by the run of computer programs. A question was then to know how to consider this heat regarding, computational processes. On this interrogation, one can distinguish two paradigms that we will refer to as the *pure-compute* and *compute-and-heat*.

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.

P.-F. Dutot and D. Trystram (Eds.): Euro-Par 2016, LNCS 9833, pp. 389–401, 2016.
DOI: 10.1007/978-3-319-43659-3_29

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 [5]. 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 production. Though heat generation was already observed in successive generation of electronic computers, it is only in recent years that this second vision was clearly set with the development of heat recovery systems in data centers [1] and in the pioneer initiative of Qarnot computing[1].

In his report on the Babbage analytical engine [7], Menabrea suggested to consider two economical gains we could expect from general purpose computers: the economy of time and the economy of intelligence. The conviction that supports this work is that in shifting from the pure-compute to the compute-and-heat paradigm, we could add another gain: the economy of energy. This is because the compute-and-heat paradigm is based on a vision where we can merge a computer and a heater into a single machine. However, this reasoning holds if we are able to establish that the compute-and-heat vision could lead to *satisfactory* solutions for heating and computing. For this purpose, our paper focuses on the implementation of the compute-and-heat paradigm made by Qarnot computing.



(a) External view (b) Some sensors

Fig. 1. The Qarnot digital heater.

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 plateform based on special *server* nodes named digital heaters or Q.rads (See Fig. 1). Each Q.rad embeds several processors connected to a heat diffusion system. Q.rads are deployed in homes, offices, schools etc. and the network of radiators that they constitute is the physical infrastructure of the distributed data center.

[1] http://www.qarnot-computing.com/.

The Qarnot model is based on a new and customized resource manager named Q.ware. In comparison with traditional ones, Q.ware supports a service provisioning model that distinguishes two types of requests: requests for heating and for computing. In addition, the requirements in computations must be balanced with those in heating. Our paper summarizes the design principles of the middleware and presents the new computational problems for which it was built. It also proposes an empirical characterization of the middleware performance on thermal comfort and processing time. These prelimary results show that Q.ware can be used to develop distributed systems that are competitive with traditional clouds and heating systems.

The remainder of this paper is organized as follows. In Sect. 2, we present and discuss the related works. Section 3 describes the Q.ware manager. An empirical characterization of performance is proposed in Sect. 4 before concluding in Sect. 5.

2 Related Works

At first sight, it might not be easy to perceive the interest in developing a new resource manager for geo-distributed computing. Indeed, the study of resource management is a well-investigated topic and several managers already demonstrated their efficiency in the management of huge computer systems. However let us notice that while these managers were mainly developed for the management of grids, clusters, or desktop grids, they are not necessarily adequate for geo-distributed clouds. Indeed, clouds are based on alternative models for service provisioning that introduce specific scheduling problems like the virtual machines consolidation and placement problem.

Several resource managers were proposed for geo-distributed clouds. In [2], 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 the processing. The requirements in machines is transmitted to an IaaS provider that will then boot the necessary compute nodes with virtual machines. The grid scheduler then deploys the job on the virtual machines that are started. The open stack project[2] developed several solutions for the creation of distributed clouds. This includes distributed scheduling mechanisms implemented in Nova or the open stack cascading solution that can be used for an hierarchical management of several open stack sites. The NebulaEdgeCloud [10] proposes a distributed edge computing. It enables distributed data-intensive computing through a number of optimizations including location-aware data and computation placement, replication, and recovery. The SlapOs cloud [11] 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 sent reports to

[2] https://www.openstack.org/.

the master about their state. 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 [3]. 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 [6]. Despite the existence of such solutions, they do not all implement one of the main feature required in the Qarnot vision, which is the need to define a thermal-aware manager.

We did not find in the literature any geo-distributed cloud resource manager whose specificity is to be thermal-aware. However, let us notice that thermal-aware schedulers were investigated on several computing systems. Thus, one might consider that a thermal manager for geo-distributed cloud would easily be derived in including existing scheduler policies in a geo-distributed manager. However, let us observe that this will not necessarily conduct to the heating as a service model that is targeted. Indeed, existing thermal-aware scheduling policies mainly targeted the minimization of the power consumption and heat generation, the minimization of the flow of hot air or the minimization of the cooling costs of computer systems [8]. This also means that they are formulated in the pure-compute paradigm while with Q.ware, it is the compute-and-heat paradigm that is followed. The consequence is that Q.ware will reduce the temperature only when requested and must manage target of temperatures. In addition, Q.ware can operate in an highly dynamic environment in which it is the requirements in heating that determines the computing power of its nodes. The next section gives a general view of the architecture of the middleware.

3 The Q.ware Resource Manager

3.1 Design Principles

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

- **Distributivity:** Q.ware assumes a physical geo-distributed topology for cloud computing. In the case of the Qarnot cloud for instance, the compute nodes consists of digital heaters deployed in buildings of the city of Paris. The management of the geo-distributivity is in particular handled by locality-aware rules in the scheduling.
- **Security:** Q.ware integrates state-of-the-art security modules for encryption and authentication. The REST API is accessible through HTTPS or leased line; All distribution and computing nodes implement TLS/IPSEC with client authentication. In addition, the compute node could be stateless (no storage).
- **Thermal-awarness:** 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.

- **City-cloud:** The Q.ware is based on a hierarchy of 3 levels of servers: Q.node, Q.box, Q.rad. The organization has been conceived to manage a data center distributed in a city. 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 city level.
- **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 BOINC (http://sat.isa.ru/pdsat/top_users.php).
- **Fault-tolerance:** 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.rad will ensure the heating supply.
- **Scalability:** We can easily add new compute nodes to Q.ware by creating new instances of the different servers.

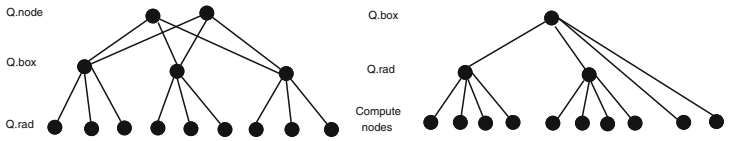
The Q.ware resource manager includes a C# API and several SDK (Python, NodeJs, C#) for the submission of computing requests. In the next section, we will present the architecture of the manager in more details.

3.2 Architecture

The Q.ware is composed of three types of servers: Q.nodes, Q.boxes and Q.rads. 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. 2(a). As Q.ware was designed to operate data centers distributed in cities, a typical deployment of the middleware could be seen as a forest of Q.nodes. 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 might not be suitable. For fault tolerance issues, it might be more interesting to link a Q.box to several Q.nodes. Ideally, Q.ware assumes that the computing power of the cloud is in the processors (we will also use the term compute nodes) inside the digital heaters. But, as already mentioned, it can also manage these resources without the digital heater abstraction (See Fig. 2(b)). In particular, Q.ware can exploit concurrently the compute nodes of a heater and container/virtual machines deployed in another data center.

In the server hierarchy of Q.ware, the heating requests are first routed towards the leaf nodes while the computing requests are routed to the top. For computing requests, Q.ware supports a Python and C# client that is used to formulate the computing requests as the submission of a set of tasks. Here, a task refers to the run of a container/virtual machine image and is associated with a set of input files, a virtual disk 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



(a) View of the servers hierarchy (b) Compute nodes, Qrads and Q.boxes

Fig. 2. Possible deployment with Q.ware.

higher priorities to deploy. Once done, several filters are applied to determine the compute nodes that will run the tasks. We will come back on the scheduling model further. For now, it is important to retain that the vision that the Q.ware has of the compute nodes come 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, in each digital heater. The Q.rads servers have a direct access to the processors, sensors and the main computing power information. The sensors include humidity, light, CO2, noises, temperature etc. 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 connexion of its Q.box. In the case, the communication is broken between the Q.box and the Q.nodes, the Q.rads are still able to compute cached tasks. In the case where the communication with the Q.box is broken, are not working, 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 Qboxes manage the storage used in the heaters. The choice to separate the management of the storage was initially driven by noise and integration consideration. 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 the processors of the heaters it controls. Those decisions are based on collected sensors data. Finally, each Q.box is connected to at least one Q.node. In the same way, a Q.rad can be connected to one or several Q.boxes. These choices were made for being fault-tolerant.

3.3 Scheduling in Q.ware

One main novelty in Q.ware is its scheduling model. To understand why, in this part, we will present here some challenges envisionned in the design of the middleware.

A New Objective Function. Scheduling in Q.ware is naturally a multi-objective problem. This is because there are at least three viewpoints: the one of customers that want to compute (*HPC customers*), the viewpoint of customers that want to be heated (*host*) and the one of the middleware.

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 C_{max} minimization. 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 i want 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 schedules 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:

$$\text{minimize} \max_{1 \leq i \leq m} \int_0^{T_m} |target_i(t) - ambiant_i(t)|.dt$$

Here, T_m is an input estimation of the time required to process the jobs. Finally, the middleware viewpoint is specifically related to one goal of the Qarnot computing business model. It is the one of reducing the energy consumption in the processing of the jobs. This means that if $P_i(t)$ is the power consumption of the heater i at date t, the objective is to minimize:

$$\max_{1 \leq i \leq m} \int_0^{T_m} P_i(t).dt$$

This objective is related to the Qarnot business model since the company refunds the electricity bill of the hosts. At first sight, it might seem impossible to reduce the energy consumption on a system that produces heat from electricity. However, we can play on the inertia of the heater to not compute all the time.

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 [4]. This is done in Q.ware where the hosts and Q.ware viewpoints are defined as constraints in an adaptive scheduling approach.

A New Scheduling Problem. As we have a new objective function, we have a new scheduling problem. But, the novelty of the scheduling problem in Q.ware is not only related to the objective function. The Q.ware scheduling model also

introduces two types of constraints related to heat production. To understand why, let us consider the power equation:

$$P = CV^2 f + P_{static} \tag{1}$$

Here, P_{static} is a base power consumption and $CV^2 f$ is the dynamic power where V is the voltage, f the frequency and C the capacitance. In Q.ware, this power equation is used to manage the heat production in Q.ware. The idea is to act on the dynamic part to consume more or less electricity and then generate heat. There are two new types of heat production constraints we consider. The first is on the *velocity* at which we go from a temperature $\Delta(t)$ to $\Delta(t')$ $(t' > t)$. Here, we define the velocity as

$$v(\Delta(t), \Delta(t')) = \frac{\Delta(t') - \Delta(t)}{t' - t}$$

The value of the velocity will depend on the configurations we will set for the load, voltage, frequency of the processors inside the heater. The velocity constraint we consider is that a host could set a minimal speed in heating. In general, all velocities cannot be reached because each heater has a limited power consumption. The second type of constraints is to ensure that the heat diffused by any heater fits within a given interval. The guarantee of a minimal and maximal heat is important since some temperatures are more suitable for the human body. This constraint implies to have an accurate model of the inertia of the heater.

A New Model for Heterogeneity. Since processors frequencies are manipulated to produce heat (see Eq. 1), scheduling in Q.ware must be envisionned in an heterogeneous context. This hypothesis holds even if *the processors layer of the distributed architecture initially have the same characteristics*. Scheduling algorithms for variable speed processors have been investigated in the past. But the main novelty that the Q.ware model introduces is that the variability is dynamic and could be caused by the interaction of the hosts with the heater (modification of the target temperature for instance). In its current implementation, Q.ware supports a simple variability model that calibrates the computational power of the nodes depending on the seasons. But a more elaborated model that uses the sensors embedded in the heaters is in construction. The idea is to to anticipate the available computational power by detecting the presence of host and combining this information with meteorogical parameters like the external temperature, the humidity etc.

An Open Perspective for Non Cooperative Game Scheduling. During the summer, the available computing power in Q.ware will decrease. To increase it, we can however turn the scheduling problem in a game where hosts compete in an ecological perspectives. Given two hosts $host_1, host_2$ let us assume that each could tolerate a small deviation over the temperature they want inside

their homes. In a ecological viewpoint, this is interesting because it allows the processing of the tasks in using a free-cooling system. Let us assume that in manipulating the target temperature, each host can either accept or reject a task that was scheduled on his heater. The deviations that $host_1$ could tolerate could differ from the one that $host_2$ could tolerate. In order to make these threshold grow, let us consider a game with four participants: the Q.ware, $host_1$, $host_2$ and a datacenter. Initially in the game, n tasks are submitted to Q.ware. At the round j of the game, the Q.ware pushes a task and chooses a host that will process it. If the host accepts the task, it cumulate the reward associated with it. If not, Q.ware will propose the task to the other host. If none of the hosts accepts, then it is the datacenter that will run the task. Such a game will put in tension two objectives: the thermal confort of each host (private interest) and the ecological benefit for all (public interest). In practice, the winner of such a competition could be the datacenter. To avoid this situation and maximize efficiency, Qarnot gives priority to optimal buildings for deployment (e.g. schools closed in summer) and plan to leverage on others opportunities to exploit heat (e.g. hot water).

Here ends the presentation of the Q.ware middleware. In the next section, we will discuss its performance.

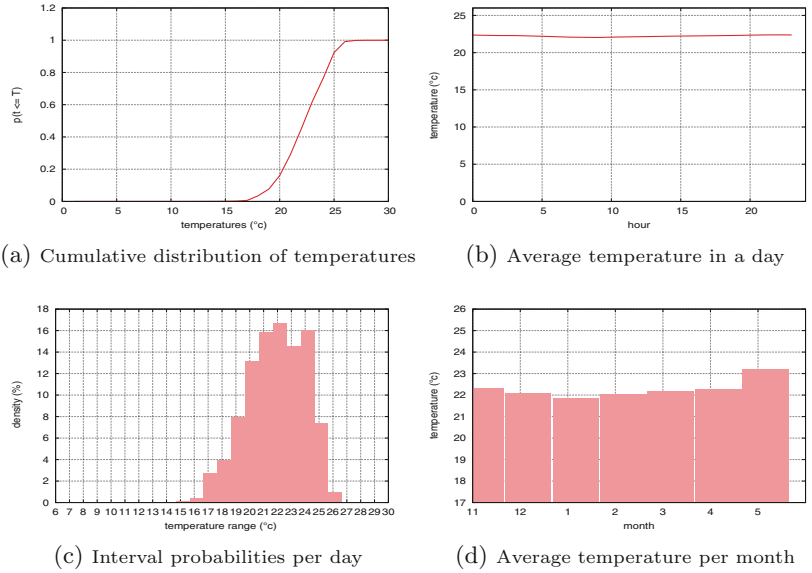
4 Performance Characterization

In this section, we propose an empirical characterization of the performance we could expect from Q.ware. The characterization is based on data collected from the Qarnot cloud. It is important to notice that it is not easy to dissociate the impact of the physical architecture of the cloud on the performance we measured. However, as Q.ware was mainly designed for the Qarnot cloud, we can say that the estimation are significant. We estimated the performance on two criteria: the thermal comfort and processing time. We start with the thermal comfort.

4.1 Thermal Comfort

On thermal comfort, we considered distribution of temperatures inside homes where the Qarnot heater was deployed. The intent was to see whether or not the system was able to produce sufficient heat to people that adopts it.

In Fig. 3(a–d), we present the trends on temperatures we observed. The figures were computed from more than 200 different heaters. The temperatures were taken from November, 01st of 2015 to May 05th of 2016. The measures were collected approximately every 10 min. The interval probabilities were computed for the intervals $[a, a + 1]$ where a is a positive integer. The curves show that the temperatures were concentrated between 17 and 26 ° with an average temperature around 22.5 °. They also show that the heat is guaranteed during the winter with a low probability to observe a temperature under 17 ° (mostly open windows). These results are interesting because as shown by a national french study [9], more than 75 % of french householders have a preference for a reference living room over or equal to 19 °. In addition, this latter study [9]



(a) Cumulative distribution of temperatures (b) Average temperature in a day

(c) Interval probabilities per day (d) Average temperature per month

Fig. 3. Temperature distributions

reveals a correlation between the increase of this reference temperature and the revenue level of the householders. We can thus conclude that the distributions we observed reflect a vision of the comfort shared by most french householders.

It is important to precise that since these measures were done with sensors embedded in the heater, there might be a biais when we go far from the heater.

4.2 Processing Time

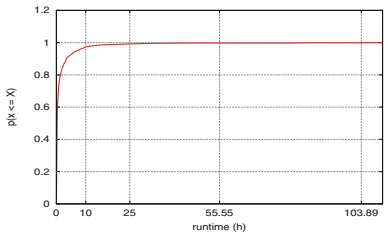
We analyzed the journal of events of the Qarnot cloud scheduler. This journal reports the state of the tasks submitted in the platform. One of these task consists of the execution of the Qarnot parallel rendering service[3]. We focused on it (more than 1500 tasks). In Fig. 4(a), we depict a cumulative distribution of the rendering tasks that were completed (some failed). This figure shows that more than 80 % of the tasks we analyzed were processed in less than 3 h (Makespan). It also shows that some of the processed tasks took more than 80 h. This is very interesting if we consider that we did not get an error even if the processors temperature in these cases often exceeded 80 °. This first curve shows that the idea of provisioning heating from computations can perfectly be applied to long-term and compute intensive tasks.

The tasks we considered were diverse. To show this point, we computed a *load balancing factor* as the ratio between the sum of completion time required for processing a task and the makespan times the number of processors used

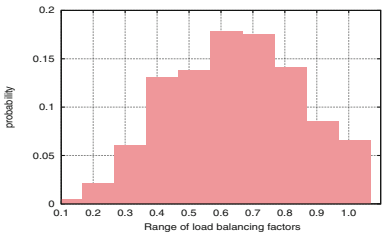
[3] blender.qarnot.net.

in the run of the task $\left(\frac{\sum_{C_i}}{P.C_{max}}\right)$. Somehow, this factor could approximate the parallel efficiency achieved on the task in the case where the sequential time is the parallel time on one processor. Figure 4(b) gives the interval probability to fall in different range of load balancing. As one can notice, there is not an interval that dominates the other.

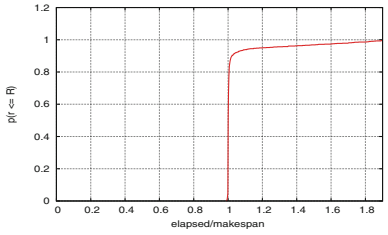
In the parallel rendering, any input file is splitted into several sub-files that each is associated with a sub-task to deploy on the heater (Bag of Task parallelism). If these subtasks are deployed on a set S of processors, we considered that the maximal time spent on each processor $p_i \in S$ is the effective computing time. For each task, we computed the ratio between the elapsed time in the processing of the task (the one perceived by HPC customers) and the effective computing time $\left(\frac{\text{elapsed time}}{C_{max}}\right)$. The goal was to capture the overhead induced by Q.ware (system deployment, remote access to the persistent storage, execution faults, variation of processors speed etc.). Figure 4(b–c) depicts the result we observed. We distinguish two cases: in the former, the number of cores used for the rendering is lower than 64 and in the latter it is greated. We made the distinction to see the impact of the number of processors on the overhead.



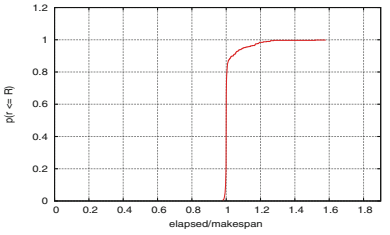
(a) Cumulative distribution of runtime



(b) Interval probability of balance factors



(c) Cumulative distribution of the overhead in the release date (1-64 cores)



(d) Cumulative distribution of the overhead in the release date (over 64 cores)

Fig. 4. Runtime distributions

As one can notice, the probability for the overhead to be close to 1 was high in all cases. This means that the time taken for the management of the resources is not predominant in Q.ware. However, let us notice that the overhead reached 1.7 in some cases. This phenomenon occurs on short jobs. For instance, the execution

of a sub-task in the parallel rendering service always includes a deployment time (operating systems, copy of files etc.). The shorter the execution time, the higher the importance of the deployment time.

5 Conclusion

In this paper, we introduced Q.ware, a new resource manager for a utility computing approach in which heating is considered as a cloud-service. We described the architecture of the middleware and proposed a performance characterization in considering data collected from the Qarnot cloud. The results showed that Q.ware could pave the way of a new vision of distributed computing where cloud-services and heating are provisioned from a unique platform. The results also invite to (1) 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.

To continue this work, we mainly envision an extensive benchmarking study of the middleware. The objective is to consider other dimensions of the quality of services like the reliability and the availability. We also intend to make a comparative analysis with the performance of other resource managers.

References

1. Alfonso, C., Giulio, P.: Cooling systems in data centers: 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)
2. Armstrong, P., et al.: Cloud scheduler: a resource manager for distributed compute clouds. CoRR abs/1007.0050 (2010)
3. Buyya, R., Barreto, D.: Multi-cloud resource provisionning 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
4. Dutot, P.F., Rzadca, K., Saule, E., Trystram, D.: Multi-objective Scheduling, Chap. 9. Chapman and Hall/CRC Press, November 2009. ISBN: 978-1420072730
5. 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)
6. Jennings, B., Stadler, R.: Resource management in clouds: survey and research challenges. J. Netw. Syst. Manage. **23**(3), 567–619 (2015)
7. Menabrea, L.F., Lovelace, A.: The analytical engine invented by charles babbage. From the Bibliothèque Universelle de Genève, No. 82, October 1842. http://www.fourmilab.ch/babbage/sketch.html
8. Moore, J., Chase, J., Ranganathan, P., Sharma, R.: Making scheduling "cool": temperature-aware workload placement in data centers. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC 2005, p. 5. USENIX Association, Berkeley (2005)

9. Penot-Antoniou, L., Zobiri, R., (directeur de la publication), X.B.: Les déterminants de la température de chauffage adoptée par les ménages. Etude et Documents No. 83, Comissariat général du développement durable, April 2013. http://developpement-durable.gouv.fr/IMG/pdf/ED83.pdf
10. 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
11. Smets-Solanes, J.P., Cérin, 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