Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 4670271, 12 pages
http://dx.doi.org/10.1155/2016/4670271

Research Article

FPGA-Aware Scheduling Strategies at Hypervisor Level in Cloud Environments

Julio Proaño Orellana,[1] Blanca Caminero,[1] Carmen Carrión,[1] Luis Tomas,[2] Selome Kostentinos Tesfatsion,[2] and Johan Tordsson[2]

[1]*Computing Systems Department, University of Castilla-La Mancha, Campus Universitario s/n, 02071 Albacete, Spain*
[2]*Department of Computing Science, Umeå University, 901 87 Umeå, Sweden*

Correspondence should be addressed to Julio Proaño Orellana; julio.proano@alu.uclm.es

Received 25 March 2016; Revised 13 May 2016; Accepted 22 May 2016

Academic Editor: Florin Pop

Current open issues regarding cloud computing include the support for nontrivial Quality of Service-related Service Level Objectives (SLOs) and reducing the energy footprint of data centers. One strategy that can contribute to both is the integration of accelerators as specialized resources within the cloud system. In particular, Field Programmable Gate Arrays (FPGAs) exhibit an excellent performance/energy consumption ratio that can be harnessed to achieve these goals. In this paper, a multilevel cloud scheduling framework is described, and several FPGA-aware node level scheduling strategies (applied at the hypervisor level) are explored and analyzed. These strategies are based on the use of a multiobjective metric aimed at providing Quality of Service (QoS) support. Results show how the proposed FPGA-aware scheduling policies increment the number of users requests serviced with their SLOs fulfilled while energy consumption is minimized. In particular, evaluation results of a use case based on a multimedia application show that the proposal can save more than 20% of the total energy compared with other baseline algorithms while a higher percentage of Service Level Agreement (SLA) is fulfilled.

1. Introduction

Nowadays, the processing computational resources have shown an important change to achieve a balance between performance and power consumption. Devices such as GPUs and FPGAs among others are integrated as specialized processing elements into cloud environments to extend the capacity of the cloud.

FPGAs are commercial off-the-shelf reconfigurable silicon devices that achieve hardware-like performance with software-like flexibility. These are facts of great interest in the context of cloud computing. Cloud mainly leverages virtualization technology (i.e., virtual machines) to manage resources of a data center. Thus, providers can share the physical resources between clients.

In this context, resource scheduling is a critical issue that can contribute to increasing the benefits of cloud platforms. On one hand, depending on how many resources are allocated to different users' requests for service, more or fewer requests can be serviced while fulfilling their SLA. This fact impacts the benefits obtained by the cloud provider. On the other hand, additional advantages can exist, such as achieving a positive effect on the data center energy consumption. The combination of both implies getting a better Return on Investment (ROI) from the infrastructure which is crucial when serving Software as a Service (SaaS) requests with QoS requirements. Typically, a SaaS user issues a request for a specific service, with particular QoS-related constraints (i.e., a deadline). The user does not need to be aware of which type and how many resources are required to get its response. The system must be able to allocate and schedule the necessary resources (both in quantity and type) to serve this request.

This paper addresses the scheduling of jobs within a cloud infrastructure which is composed of heterogeneous physical nodes (with and without FPGA accelerators) from a hierarchical point of view. In this structure the two levels of scheduling are

(1) the Cluster Level Scheduler (CLS), to decide the type of node where the virtual machine (VM) that will serve the request will be deployed,

(2) the Node Level Scheduler (NLS), to decide which one of the VMs allocated to the node will get the use of the accelerators (FPGAs in particular) available in a node.

The problem of high-level scheduling among physical nodes, referred to as CLS, has previously been addressed by the authors in [1, 2]. Now, this work focuses on the impact of different scheduling strategies within a physical node with FPGA resources, which are applied at the hypervisor level. In a nutshell, the main contributions of the paper are the following:

(i) to extend the hypervisor functionality to support dynamic control of FPGA devices as cloud computational resources,

(ii) to propose a novel FPGA-aware Node Level Scheduler (NLS) metric aimed at QoS provision while reducing energy consumption,

(iii) to present a novel fine-grained FPGA-aware Node Level Scheduler,

(iv) to evaluate the Node Level Scheduler proposals in a real testbed, comparing them to some simple scheduler techniques.

This paper is structured as follows. Sections 1 and 2 introduce and motivate the problem being tackled. Section 3 reviews literature related to the topic of the paper. The framework where the presented research has been carried out is outlined in Section 4. Next, Section 5 provides the details on how accelerators (and in particular, FPGAs) are included into the scheduling strategies, and the additional fine-grained level of scheduling is introduced in Section 6. The evaluation of results are presented in Section 7. Finally, Section 8 provides some concluding remarks.

2. Background and Motivation

During the last few years, cloud computing has consolidated as a paradigm that enables a flexible and on-demand use of IT resources at different levels (infrastructure, platform, or software) as a service. Typical cloud providers support their service by means of massive data centers (usually spread around the globe), while cloud users get access to the resources with a pay-per-use model. This is referred to as a public cloud model. The cloud computing paradigm can also be applied within an organization, leading to the private cloud deployment model. Resources are pooled and shared among the different organization user groups to meet their demands on IT resources. In either case (public or private), cloud computing platforms are composed of pools of computing, storage, and networking resources that are made available as a service to their users. Service Level Agreements (SLAs) are established between providers and users, to specify the conditions of the service, both from technical (availability, Quality of Service, ...) and formal (cost of the service, penalties in case of contract breach, ...) points of view.

Many challenges exist in this model that turn out to be the focus of many research projects, such as efficient resource management [3], security and privacy concerns [4], or standardization [5], just to name a few of them. In particular, some kind of orchestration is needed in order to allocate the adequate resources to every user request [6]. As it has been pointed out before, cloud providers face the following dilemma: admitting more users into the system would lead to more incomes, but if the available resources are not enough for fulfilling the established SLAs, penalizations will cut benefits down. Thus, care must be taken when admitting more users into the system.

From another point of view, data center's energy consumption is nowadays a big concern. Energy costs are a major contributor to a data center's Total Cost of Ownership (TCO) [7]. Thus, strategies to improve data center's efficiency are the topic of many research efforts, such as adjusting the voltage supplied to servers according to their workload [8] or even creating specific hardware designs [9]. One strategy that can provide benefits regarding energy consumption as well as performance in data centers is the integration of accelerators as specialized resources within the cloud infrastructure [10], such as General Purpose Graphics Processing Units (GPGPUs) or Field Programmable Gate Arrays (FPGAs).

FPGAs allow cloud computing data centers to scale up in a more efficient way than using just conventional processors [11]. It is interesting to note that the initiatives towards building exascale systems with low-energy consumption driven by the European Union (EU) are based on the integration of FPGAs with ARM processors [12].

However, using heterogeneous resources in cloud environments is still an open challenge, as illustrated by the work presented by Microsoft in [13]. Frequently, cloud applications which are running on VMs with a fixed amount of hardware resources including accelerators do not use the same number of them all the time due to their features. Thus, a proper scheduling strategy might help to optimize the utilization of these resources.

Moreover, different applications could benefit from the FPGA over time, depending on their degree of compliance with QoS requirements. To achieve this objective, the FPGA should be properly shared among competing service requests.

3. Related Work

Incorporation of FPGAs in the cloud context is a relatively new area of research. So, the Catapult project [13] led by Microsoft represents the first detailed investigation of applying FPGAs within an enterprise-level data center application. In this case, FPGA-accelerated nodes are used for the Bing web search engine. Results show great improvements in both the latency and the throughput of the service.

Nevertheless, FPGA-aware scheduling in a cloud environment is still at initial stages. Integrating FPGAs as first class computational resources to provide cloud service demands novel high-level programming models to simplify the development of software applications, maximizing communication to FPGAs, and the development of efficient FPGA-aware scheduling algorithms.

One crucial step to efficiently run applications across heterogeneous hardware is to provide optimized versions of computational kernels (as BLAS routines or FFT) [14]. OpenCL [15], VIVADO [16], and Lime [17] are high-level programming models focused on reducing the time-to-market in the designing process.

In addition, providing FPGAs as cloud computing resources demands virtualization support in hardware with

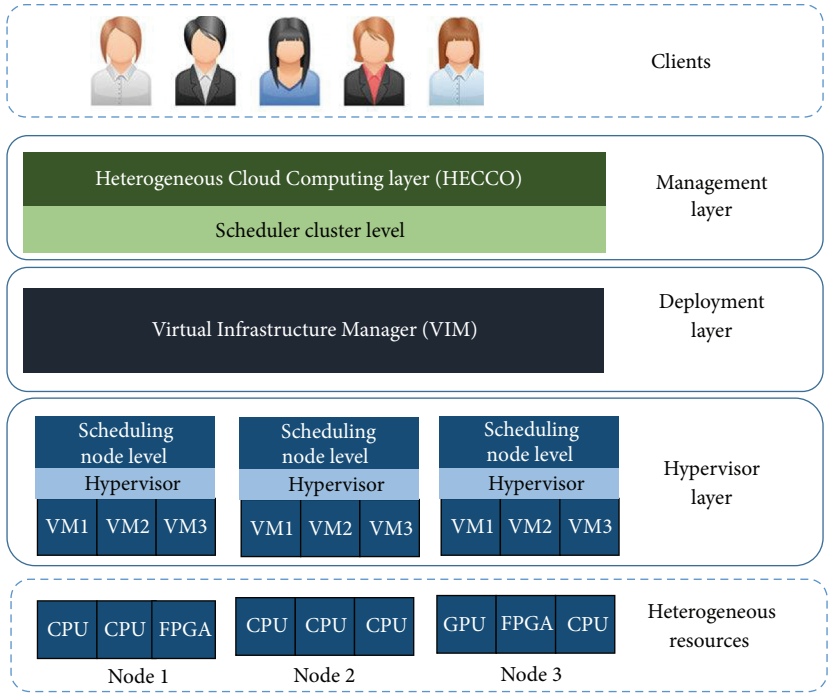

Figure 1: Architecture overview.

near-native performance. Open source interface frameworks have recently emerged (RIFFA [18], DyRACT [19]) that enable FPGA designs to be accessed through an abstracted software API on the host with communication throughput between the host and FPGA close to the limits of modern PCIe interfaces.

Moreover, efficient resource managements and scheduling algorithms are open key challenges to providing FPGA cloud services. The scheduling of jobs faces a multiobjective optimization problem in the process of assigning jobs to a pool of limited resources. Resource management and scheduling have been a hot research topic in a cloud context, and many approaches have been proposed to place the VMs to physical CPU hosts. A survey of the most recent proposals about metaheuristic scheduling solutions for cloud can be found in [20].

In this context, cloud-centric integration of FPGAs frameworks is developed in [11, 21] by extending the open source cloud management system OpenStack [22]. The systems support the on-demand deployment of a user designed custom hardware while maintaining the cloud computing benefits of scalability and flexibility. The bitstream of the application is considered as a special VM image. So, FPGAs are eligible as computational resources by the clients.

Another recent related work presents a framework that integrates FPGAs in a standard server with virtualized resource management and communication [23]. The resource management selects the smallest reconfigurable FPGA area able to attend the request. If no one is found, the user request is rejected, and the request can be processed in software. As an application case study, they built a MapReduce accelerator for word counting and preliminary results are evaluated about integrating FPGA devices in the cloud using partial reconfiguration.

At some points, the systems mentioned above can be complementary to the work presented in this paper because all of them focusing on supporting FPGAs as computational cloud devices. But in contrast, in our case FPGAs are not visible to the clients. The idea is to provide a QoS Software as a Service by making efficient use of FPGA-aware scheduling algorithms. Our work focuses on developing a cloud management framework able to scale up and down FPGA devices in a dynamical way.

4. Hierarchical Scheduler Architecture Overview

In previous works [1, 2] the problem of integrating FPGAs into a cloud was faced by "HECCO" (Heterogeneous Cloud Computing) architecture. In those approaches, every node of the cloud system is composed of one or more CPUs with a certain number of cores each and zero or more accelerators. Frequently, the number of accelerators available within a node is lower than the number of CPU cores available. Therefore, they must be shared between clients. HECCO enables the provision of IaaS and SaaS services with Quality of Service (QoS) requirements.

In this work, the proposal architecture leveraged the use of accelerators to (a) finish job tasks within a particular time frame in order to fulfill their QoS requirements, expressed as Service Level Objectives (SLO) within a Service Level Agreement (SLA), and (b) reduce the energy footprint of cloud data centers.

The whole picture of the framework is depicted in Figure 1. It is organized into several layers:

(i) The *Management Layer* is responsible for receiving client requests and selecting the most suitable node

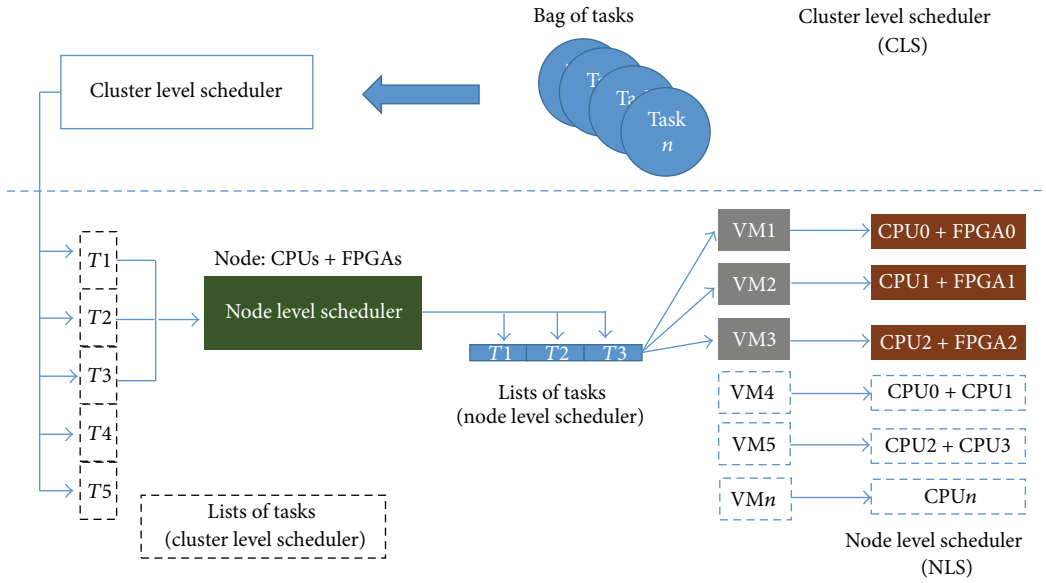

Figure 2: Different scheduling levels.

to allocate their applications. Each request is defined by a template in which parameters such as the type of service and the utilization time are defined. This layer is composed of the Heterogeneous Cloud Computing (HECCO), which includes the Cluster Level Scheduler (CLS).

As outlined above, it provides the "intelligence" to fulfill the QoS requirements expressed by clients within their SLAs. In particular, the CLS implements a series of mechanisms that are mainly aimed at allocating every request to the most appropriate resource (i.e., a node with or without accelerators). Its decision is based on the client's SLA and the availability of the resources.

More details on the management layer can be found in [2].

(ii) The *Deployment Layer*, which receives instructions from the management layer, is responsible for performing actions such as the creation and monitoring of the whole virtual environment. It has a complete view of the cloud and is composed of the Virtual Infrastructure Manager (VIM). The VIM is a centralized data center manager that allows to build and manage the cloud environment.

(iii) The *Hypervisor Layer* provides the control of the physical resources. This layer has a local view of the system and is composed of a hypervisor, which is responsible for the supervision of the VMs and resources for each node. This layer also contains a *Node Level Scheduler* (NLS) that is responsible for sharing the local resources within a node. In this case, the scheduling's decision is focused on the management of the use of the accelerators within each node, in order to get the most out of them.

More details about this element are provided in Section 5.

Figure 2 depicts the two different levels of scheduling. First, the CLS receives the tasks and selects the most suitable node to allocate each of them, following the strategies proposed in [2]. Second, the NLS manages the tasks allocated to a specific node.

More precisely, it selects the task which is going to receive the accelerator and for how long, by certain metrics.

In other words, given the fact that there may be more virtual machines allocated to a node than accelerators available, which virtual machine will deserve the use of an accelerator? In the rest of this work, some insight will be given on this issue.

5. Node Level Scheduling

In this section we will focus on the dynamic scheduling strategy for allocated jobs within a node and timing-related indicators, such as a deadline. This deadline is related to when the job must have finished in order to deem its SLA as fulfilled.

The main objective of the node level scheduling strategy is twofold: first, to meet application SLOs and second, to consume as less energy as possible. To achieve this multiobjective, the scheduling algorithm is aware of the properties of the physical nodes and due to the good performance and low-cost energy offered by the FPGAs, the scheduling maximizes their utilization.

Basically, the algorithm uses a heuristic process for assigning the physical resources inside the node (FPGAs and CPUs) to the virtual machines that will be deployed at each point of time. It is worth to mention that the use of this simple algorithm avoids the overhead of the system. Recall that the NLS is implemented within the hypervisor and should be able to run with minimal computational resources.

Require: Task_i, $i = 1, 2, \ldots, K$: Task T_i to be executed in the local node
$\text{List}_{\text{Task}}$ = Set of tasks
metric(Task_i): function which computes the metric for task Task_i
include(Task_i, $\text{List}_{\text{Task}}$): function which includes Task_i in $\text{List}_{\text{Task}}$
rank($\text{List}_{\text{Task}}$): function which sorts $\text{List}_{\text{Task}}$ according to the selected metric
top($\text{List}_{\text{Task}}$): function which extracts the first VM from $\text{List}_{\text{Task}}$

```
(1)  loop
(2)    Wait until FPGA_status == Free
(3)    for each Task_i/i ∈ 1 ··· N do
(4)      Task_i.metric = metric(Task_i)
(5)      include(Task_i, List_Task)
(6)      rank(List_Task)
(7)    end for
(8)    VM_first = top(List_Task)
(9)    attach FPGA to VM_first
(10) end loop
```

Algorithm 1: Node level scheduling algorithm.

In other words, a matching is done between computational resources and tasks (deployed as VMs) minimizing the computational impact. On one side, the scheduler algorithm has as input a list of tasks ($T1, T2, \ldots$), assigned by the CLS algorithm, with all the nonattended job requests that have been received in the node (see Figure 2). On the other side, each time a computational virtual resource is released in the node, the scheduler is notified and a matching process is done. Then, a resource driven algorithm selects the best candidates to match resources according to some efficient metric. This scheduling is called a *coarse-grained scheduling* because FPGA resources are busy (not available for allocation) until the end of an assigned task.

In addition, in this paper, a multiobjective metric is proposed and computed as the fraction of the computational work of a task and the deadline or time available to finish it up in order to fulfill the SLA.

Therefore, the multiobjective metric measures the computational workload demanded on the node over the time; that is, it provides a measure of the computational stress. And it is a fact that the higher the computational requirements, the more the energy consumption. That is why the task with most demanding requirements is matched to an FPGA resource. In other words, the algorithm matches the most demanding job according to this criterion to the VM with FPGA accelerators.

The computational work parameter involved in the metric reflects the workload that must process the VM. Then, we need to apply some criteria to calculate this value. For example, in our experiments with multimedia applications this parameter is computed as the number of video frames to be processed. Other examples are data-encryption services, where this parameter could be the size of the data that will be encrypted, or social network analysis services, where it could be the number of tweets stored in a log file. Techniques such as application profiling and using data from previous executions are frequently used to estimate this value.

The other parameter used in the metric is the deadline that is established by the client in the SLA.

To sum up, the node level scheduling process works as follows (see Algorithm 1): First, the selected metric is computed for each task. Next, the scheduler sorts all tasks based on that metric and the FPGA is assigned to the most demanding task. This process is repeated every time the FPGA becomes available. To achieve this goal, the status of the FPGA is periodically monitored. This synchronization process is an important issue in the overall scheduling process. Thus, some implementation details will be explained in the next subsection. The algorithm is aware of the heterogeneity of the node, because the most demanding task is executed in a VM deployed with FPGA resources.

5.1. Implementation Details. FPGAs are physically plugged into computing nodes via the PCI express bus. The communication between VMs and the FPGAs is made through the Intel Virtualization Technology for Directed I/O (VT-d) [24], which is an extension of the Intel Virtualization Technology. It allows a direct exchange of data between I/O devices and VMs.

As pointed out above, a strategy able to control the communication between the Node Level Scheduler (NLS) and the VMs currently running in the node is required. So, an FPGA synchronization service has been developed which consists of an exchange of status messages between the NLS and every VM deployed in the node. The full node information is wrapped into a "JSON" [25] data structure because it is easy to parse. The NLS is implemented as a daemon. It is constantly listening to the status information from the VMs currently deployed at the node. Each VM also has a daemon, which is periodically monitoring the status of the FPGA (whether an FPGA is attached to the VM or not). Besides, the VMs periodically send their identification and the progress of the running applications through an UDP socket connection to the NLS. If a VM is using an FPGA, its status is also forwarded to the scheduler by means of updating a "status file."

Figure 3 depicts the process of reassigning the FPGA to different VMs. In this example, the FPGA is initially attached

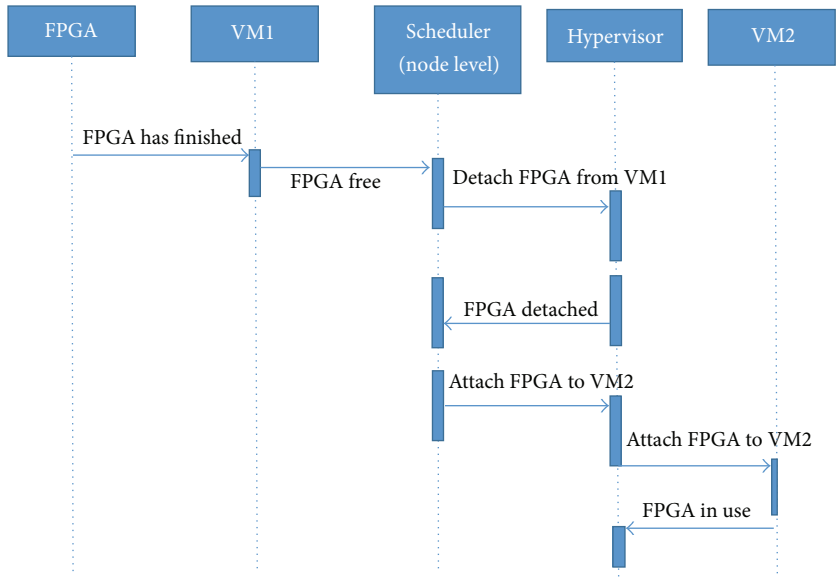

Figure 3: FPGA synchronization service example.

to VM1. Then, when computation on the FPGA ends, VM1 updates the status of the FPGA as "free." As a result, the NLS learns this state and gets aware of the fact that the reassignment process can be made safely. At this point the NLS can detach the FPGA from the current VM and attach it to another VM (i.e., the one in the head of the list of pending requests) through the hypervisor. The VM which finds an FPGA attached to it (VM2, in this case) immediately uses it and updates its status accordingly.

We use KVM [26] as hypervisor. It enables the management of the FPGA because it supports hot-plug devices with VT-d. So, it is a must that the underlying hardware supports VT-d. The NLS uses the "device_add" [27] command to attach the FPGA to a particular VM. Also, when the FPGA has fulfilled a request, the NLS uses "device_del" [27] to release the FPGA.

The NLS is not only focused on the FPGA assignment, it also considers the CPUs. In this case, the NLS matches VMs to physical CPUs using First-Fit criteria. Moreover, the scheduler uses the "taskset" [28] linux command to set the CPU affinity of a running process given its identification (PID). So, each VM has a PID and the CPU affinity "bonds" this process to the first available CPU. This is made in order to avoid swapping when a VM is assigned to different cores along time. Finally, due to the complexity of coding FPGAs, the RIFFA framework offered by Jacobsen and Kastner in [18] has been used to develop the hardware acceleration design.

In Section 7.3 the evaluation of this strategy is shown.

6. Fine-Grained Strategy

In the previous section, the Node Level Scheduler decides to assign the FPGA to a VM and the FPGA is not released until the end of the task running in that VM. But it might be the case that the task does not need to be accelerated during its whole runtime in order to fulfill its deadline. Moreover, it might occur that other tasks (which initially was not placed at the head of the list due to the value of its metric) would depend on the use of the FPGA to fulfill its deadline. Thus, reconsidering the allocation of the FPGA with a finer granularity could improve the number of fulfilled SLAs and, consequently, improve the system ROI. Under this context, a strategy which consists of dividing each task into smaller subtasks (called "chunks") has been explored. A trade-off between the execution time of a chunk and the frequency of checkpoints must be taken into account to decide the size of a chunk. The rationale is to reschedule the FPGA more often, so that more tasks can potentially benefit from the acceleration and power efficiency of the FPGA.

More precisely, all the chunks have the same requirements of computational workload, but different deadline restrictions. Figure 4 shows how a task with size S and deadline D is divided into three chunks, with sizes s_1, s_2, s_3 and deadline D. Chunk size is set according to the characteristics of the application that implements the task. This can be a certain amount of data to be processed or any other application relevant parameter. As an example, for the video processing service, the chunk size is set as a certain number of video frames to process.

Thus, Node Level Scheduler decisions on the use of the FPGA are taken every time and a VM finishes the processing of a chunk (see check point in Figure 4). In the current implementation, this is done every minute. The Node Level Scheduler selects the best candidate VM and attaches the FPGA to it as is depicted in Figure 5. The process is similar as in the previous case but now communication and scheduling are done at the chunk level. It is worth to notice that the application is completely independent and parallelizable. What it means is that the application receives a group of data as input and when the task has been finished the system sends the result as output. For acceleration process a wrapper function is used to map the application that will use CPU and

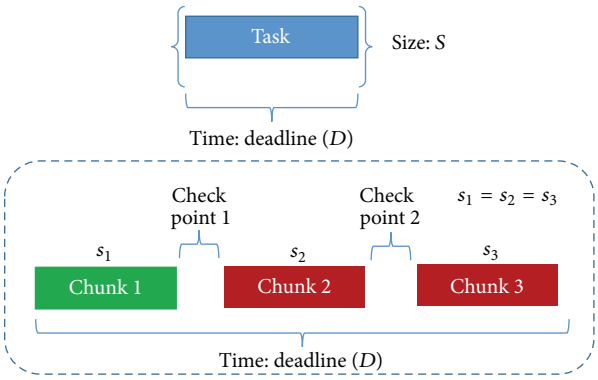

Figure 4: Division of a task to chunks.

FPGA at the same time because it is different than the one who is only running over a CPU. All the VMs of a node can take advantages of the FPGA's features and the whole node can keep the performance while the energy consumption is reduced.

In Section 7.3, some experiments are made showing the necessity to adopt a balanced criteria for all requests due to the fact that FIFO leads to lack of efficiency.

7. Evaluation

To evaluate the impact of the scheduling algorithms proposed in this paper some experiments have been carried out on a Heterogeneous Cloud Computing testbed environment, with a real application case study. Details on the platform, application, and workload are given next, prior to explaining some evaluation results.

All the experiments have been run on a real testbed, depicted in Figure 6.

Hardware is composed of an Intel Core i5 with 6 Gb of DRAM and an FPGA Virtex 6 by Xilinx (a ML605 Evaluation Kit, based on the powerful XC6VLX-240T-1FFG1156 [29]).

Additionally, as a way to measure the energy the testbed includes a power consumption monitor node. More precisely, the WattsUp PRO [30] power meter has been used. This device aims to provide an independent managed and accessed power data collecting mechanism. This device must be positioned between the computing node power supply and the main power plug, as shown in Figure 6. The output data can be recorded into a file according to a predefined interval or when particular events occur (i.e., when the power consumption exceeds a threshold previously configured). In our experiments, the monitoring frequency has been setup to one sample per second.

7.1. Application. As real use case application an Image Convolution Software (ICS) is used as a service provided by cloud. This application consists on convolving an image with a filter or kernel (integer value) in both directions horizontal and vertical. This technique can be applied also to process sequences of video. Different type of filters can be used depending on the target. For our experiments we use a Sobel filter to process a sequence of video. This video is an AVI file with resolution 720 × 384. The application has two versions, the first one runs over a CPU and the another one over a combination between CPU and FPGA.

7.2. Workload. In order to generate a realistic cloud workload we have run the real ICS application and monitored its behavior. Thus, for every test, a bag of tasks (20) was created. Each of these requests are defined by two parameters: the deadline of the task and the number of videos to process (which relates to the amount of data to be processed by the task). There are two types of service requests, depending on how demanding their deadline. The first type of requests is based on more relaxed deadline requirements (referred to as soft requirements, SR), while the second type of requests exhibits a more demanding deadline (referred to as high requirements, HR).

The HR requests are more challenging than SR because they require to fulfill stricter deadlines and specific resources such as FPGAs to ensure the QoS-related SLOs.

To set up the deadline of each service request, we have previously characterized the application. Basically, the Sobel filter application has been run on the cloud using different computational resources for different video sequences. In this step, the number of frames, the power consumption, and the time have been stored. Thus, we have got the profile of each video stream. Then, the deadline is assigned to each input request as a random value between the execution times obtained in the profile with a margin of tolerance. For instance, for a video stream composed of three chunks the deadline could be a random value between 450 and 480 seconds while for nine chunks, the values are between 1400 and 1430.

To emulate the behavior of cloud clients, the input requests rate follows a Poisson distribution $\lambda = 1/t_{\text{interval}}$. The t_{interval} is the arrival time of the next requests and it was tuned for this concrete scenario to avoid an early saturation.

Moreover, three types of bags of tasks have been created, depending on the ratio between SR and HR requests. Note that the same input request rate has been used for the different types of bags of task, only the deadline of the requests has been modified, in order to get a fair comparison of the scheduling algorithms under evaluation. In particular, workloads include 10%, 25%, and 50% of HR requests.

7.3. Evaluation for Coarse-Grained Scheduling. To carry out a performance evaluation of the novel multiobjective metric proposed in Section 5 hereinafter called Highest-Job-First (HJF), QoS compliance and energy consumption are taken into account. Moreover, for the sake of comparison both a random (RND) (with none consideration for QoS requirements) and an Earliest-Deadline-First (EDF) scheduling algorithm (the VM with closest deadline gets the FPGA use, independently of the amount of data to be processed) have also been implemented.

In order to understand the behavior of the whole system we have selected several groups of metrics.

The first one describes the behavior of the whole system. It is composed of the total energy used for the system to process a bag of tasks (energy-to-solution [31]).

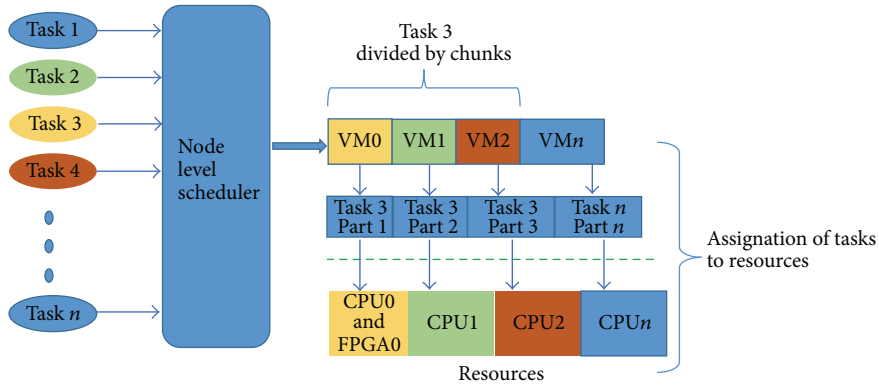

Figure 5: Scheduler by chunks.

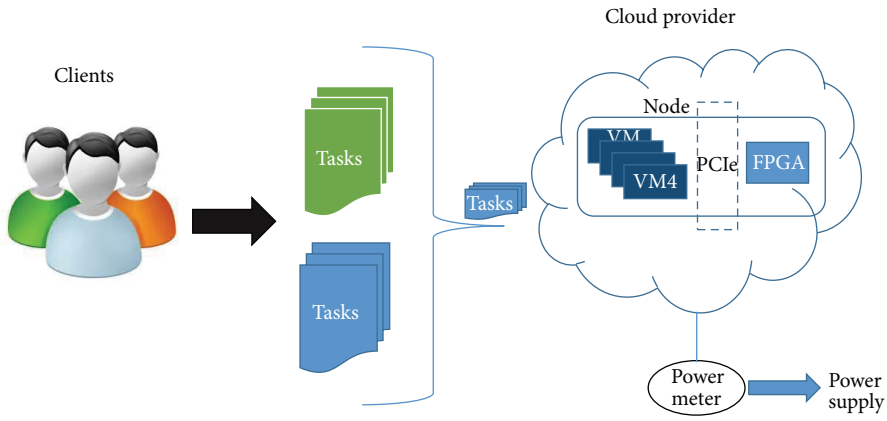

Figure 6: Cloud computing experimental testbed.

Table 1: Energy (KJoules) for different scheduling criteria.

% HR request	FPGA assignation criteria		
	RND	EDF	HJF
10%	648	638	614
25%	653	638	616
50%	657	639	634

Table 2: Percentage of fulfilled SLAs for different scheduling criteria.

% HR requests	FPGA assignation criteria		
	RND	EDF	HJF
10%	85%	85%	90%
25%	75%	80%	85%
50%	60%	65%	70%

The second group is relative to SLA compliance, the percentage of SLA fulfilled successfully, and the average energy invested per fulfilled SLA.

And the last one is focused on application performance, namely, the average number of frames successfully processed per unit of time (fps).

Table 1 shows energy consumed by the system. Results show that the HJF and EDF metrics reduce the energy-to-solution for all the type of bags of tasks with respect to the RND baseline algorithm.

Hence, from these results we can point out that the FPGA-aware Node Level Scheduler should use a QoS-aware metric.

Nevertheless, before getting a clue time an energy-to-solution should be analyzed together with the number of SLAs fulfilled. Table 2 shows the percentage of fulfilled SLAs for the different scheduling techniques. Results show that HJF gets more fulfilled SLA for all the input workloads.

Figure 7 shows the cost per SLA measured as the energy consumed. In all the cases, the cost per SLA increases with the percentage of high requirements. These result from the fact that some unlucky scheduling decisions can affect in a quite negative way the future tasks. The allocation of the FPGA to the most demanding task will change the state of the FPGA to nonfree for a period of time. Then, if the physical node receives a quite demanding task over this period of time, as no preemption of task is possible, the new request only can run in CPU resources. This can lead to a nonfulfilled SLA. Finally, regarding the performance as is shown in Figure 8 the amount of frames per second keeps an acceptable level even when more demanding requests are processed. So, in the next

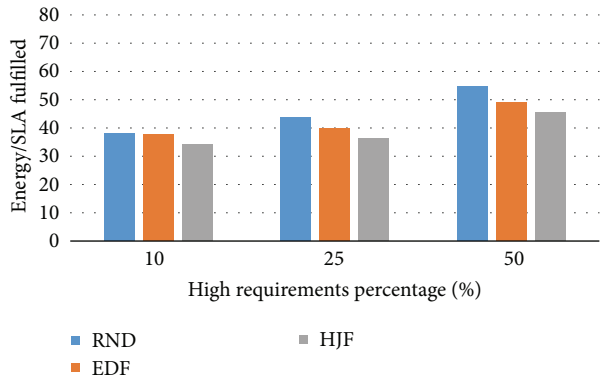

Figure 7: Energy (KJoules) per number of fulfilled SLAs.

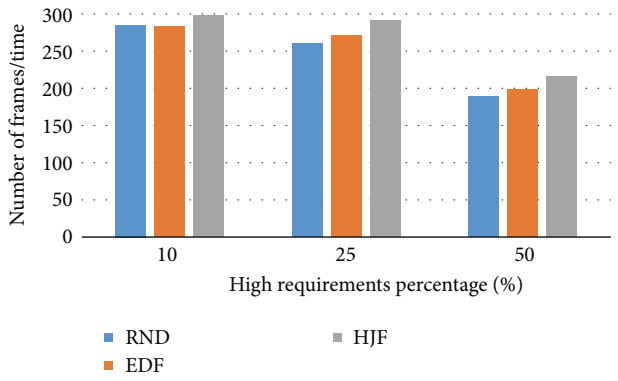

Figure 8: Number of successfully processed frames per second.

section we will evaluate our solution (fine-grained scheduling see Section 6) to address this problem.

7.4. Evaluation for Fine-Grained Scheduling. In this section the fine-grained scheduling strategy proposed in Section 6 will be evaluated. Evaluation conditions are the same as detailed in Section 7. Recall that now the task is subdivided into chunks and the FPGA assignation mechanism is job-aware.

Table 3 shows that the energy tends to rise with the percentage of high requirements requests (these metrics present the same behavior as in previous evaluation). However, the energy per SLA fulfilled decreases 35% for the HJF scheduling strategy in comparison with the other strategies (RND, EDF) (see Figure 9). Results also show that the energy relative to the number of fulfilled SLAs tends to decrease which means a save of effective energy. Thus, the combination of both an FPGA as an accelerator and an efficient scheduling strategy allows the system to save energy. The reason is the system's selection of the most suitable resources for each request and the speedup of the FPGA with lower energy impact. On the other hand, the percentage of fulfilled SLAs (see Table 4) shows a slightly upward trend (20%) for the HJF scheduling with 50% of high requirements. What it means is that the system is able to keep an acceptable ratio of SLA fulfilled.

Figure 10 shows the performance of the system in terms of the number of processed frames per second. The use of

Table 3: Energy (KJoules) for different scheduling criteria (fine-grained).

% HR requests	FPGA assignation criteria		
	RND	EDF	HJF
10%	686	665	626
25%	694	699	646
50%	774	772	696

Table 4: Percentage of SLAs fulfilled for different FPGA assignation criteria (fine-grained).

% HR requests	FPGA assignation criteria		
	RND	EDF	HJF
10%	90%	95%	95%
25%	80%	80%	85%
50%	55%	70%	75%

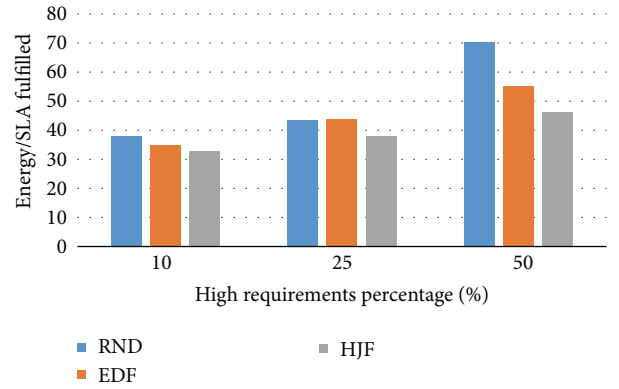

Figure 9: Energy (KJoules) per number of fulfilled SLAs (fine-grained).

the proposed HJF strategy clearly outperforms the RND and obtains similar results to the EDF. However, as shown before, this is achieved with a lower energy consumption for the HJF metric.

7.5. Fine-Grained Scheduling Optimization. The last strategy presented above raises the following concern: when we check with a certain frequency which VM needs most the FPGA in order to fulfill its requirements, why not to check also if the deadline is past? In case the deadline of the task had arrived, the VM would be killed. In this way, no task keeps running after its deadline, savings on energy.

When a SLA is violated, apart from penalties, some amount of energy is wasted because the system uses the resources to run a task that will not successfully end before its deadline. The system behavior shows that keeping VMs alive has a direct effect on the total energy and number of SLA fulfilled. Now, an optimized scheduling, where VMs running a request whose SLA has been violated are killed, is analyzed.

In this scenario, even though the scheduling and communication issues are the same as before, energy consumption for all the FPGA assignation strategies is reduced as shown in Table 5.

Table 5: Energy (KJoules) for different scheduling criteria (optimized fine-grained).

% HR requests	FPGA assignation criteria		
	RND	EDF	HJF
10%	678	649	597
25%	638	619	611
50%	637	563	586

Table 6: Percentage of SLAs fulfilled for different FPGA assignation criteria (optimized fine-grained).

% HR requests	FPGA assignation criteria		
	RND	EDF	HJF
10%	90%	95%	100%
25%	80%	85%	90%
50%	65%	70%	85%

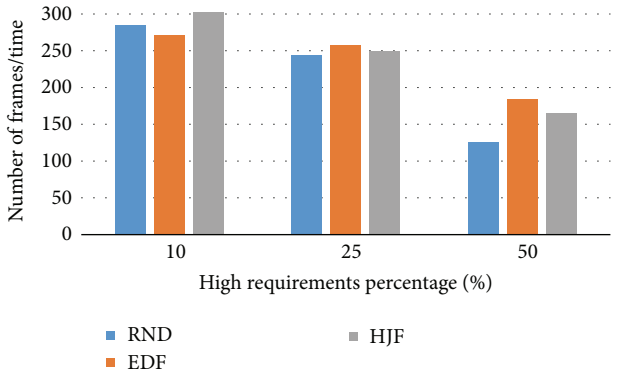

Figure 10: Number of successfully processed frames per second (fine-grained).

On the other hand, if we consider the effective energy per fulfilled SLA, 40% of consumed energy is saved (see Figure 11). In addition, the percentage of successfully fulfilled SLAs is 15% better for HJF (see Table 6). Finally, Figure 12 shows an increase of 20% of frames per second for HJF in the most demanding scenario (50% of high requirements).

To sum up, releasing the resources involved in the execution of a task when its deadline is due improves the percentage of fulfilled SLAs and increases the performance of the cloud system while reducing the energy waste. A comparison summary of the node level scheduling impact is presented in Table 7.

Figure 13 shows the number of frames unsuccessfully processed due to the SLA violation occurring. However, the number of frames unsuccessfully processed decreases significantly for the optimized fine-grained strategy because it allows taking advantage of killing the VMs when their deadline is achieved.

7.6. Evaluation Summary. Table 7 summarizes the different results when applying the different FPGA assignation strategies and scheduling techniques. It can be seen that

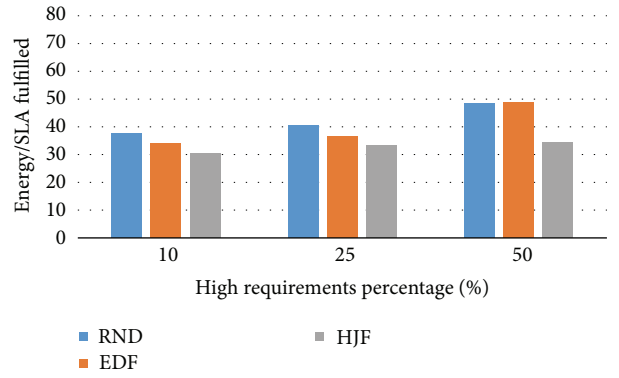

Figure 11: Energy (KJoules) per number of SLAs fulfilled (optimized fine-grained).

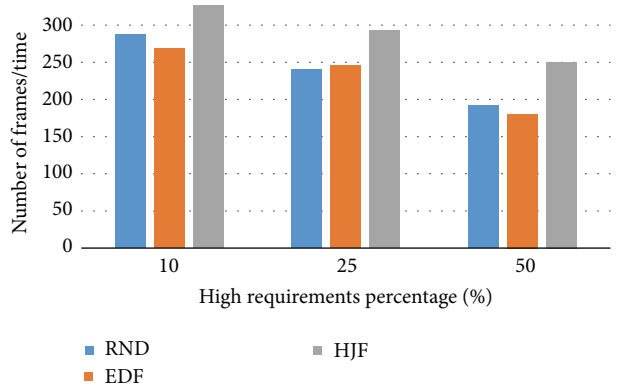

Figure 12: Number of successfully processed frames per second (optimized fine-grained).

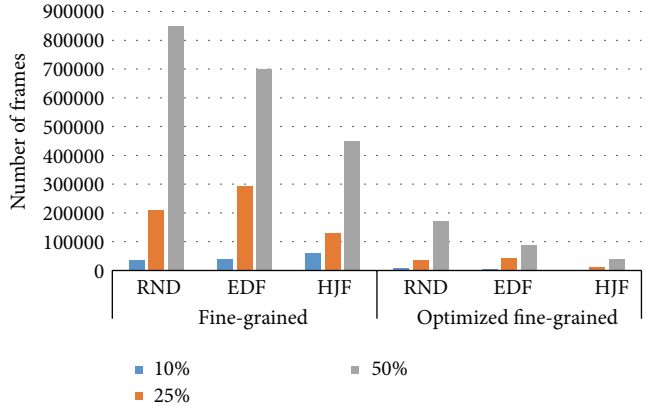

Figure 13: Number of processed frames belonging to unsuccessful SLAs (wasted).

for all the cases, the proposed HJF policy obtains the best performance, in terms of SLA fulfillment and energy consumption. Moreover, the configuration that yields the best results is combining HJF policy with the optimized fine-grained assignation strategy. Recall that for "fulfilled SLAs" and "Frames processed per second," the greater is better, while for "Energy/fulfilled SLA" the lower is better.

Table 7: Summary of tests.

% of high requirements	Scheduling strategy	FPGA assignation criteria	% of fulfilled SLAs	Energy/fulfilled SLA	Frames processed per second
10%	Coarse-grained	RND	85	38	285
		EDF	85	37	283
		HJF	90	34	300
	Fine-grained	RND	90	38	284
		EDF	95	35	271
		HJF	95	32	302
	Optimized fine-grained	RND	90	37	286
		EDF	95	34	268
		HJF	*100*	*30*	*326*
25%	Coarse-grained	RND	75	43	260
		EDF	80	39	271
		HJF	85	36	291
	Fine-grained	RND	80	43	244
		EDF	80	43	257
		HJF	85	38	249
	Optimized fine-grained	RND	80	48	240
		EDF	85	48	245
		HJF	*90*	*34*	*292*
50%	Coarse-grained	RND	60	54	188
		EDF	65	49	198
		HJF	70	45	215
	Fine-grained	RND	55	70	126
		EDF	70	55	184
		HJF	75	46	164
	Optimized fine-grained	RND	65	48	192
		EDF	70	48	180
		HJF	*85*	*34*	*250*

8. Conclusions and Future Work

We have presented a hierarchical scheduler framework to manage heterogeneous resources within a SaaS cloud environment. This framework is responsible for selecting on-demand the most suitable resources for a service while keeping a balance between performance and power consumption. The key of the framework is to maximize the number of fulfilled SLAs saving energy by making efficient use of FPGA devices. Thus, we have proposed a novel dynamic scheduling metric which considers a combination of the workload of a task, its remaining time to complete the task, and the deadline. In addition, a fine-grained accelerator-aware scheduling has been developed and improved by releasing the resources associated with a task as soon as the system realizes its SLA is not going to be fulfilled.

The proposed techniques have been compared with some well-known scheduling strategies such as Earliest-Deadline-First and Random. Experiments carried out over a real testbed indicate that the proposed metric together with the optimized fine-grained scheduling strategy increases the performance of the system even when more demanding requirements are involved. Moreover, these techniques save 23% of the total energy while the percentage of fulfilled SLAs is 85% under the most demanding workload conditions. This can be explained by the fact that FPGAs (as a accelerators) have a great ratio performance/power consumption for certain type of applications. Thus, a proper use of these devices can be turned into benefits for clients and providers.

The efficiency of this approach can be further improved by using more sophisticated strategies such as machine learning algorithms. Also, as a future work we will include the memory and network metrics as variables to optimize the use of the computational resources.

Competing Interests

The authors declare that there are no competing interests regarding the publication of this paper.

Acknowledgments

This work was supported by the Spanish Government under Grant TIN2015-66972-C5-2-R (MINECO/FEDER) and by Ecuadorian Government under the SENESCYT Scholarships Project.

References

[1] J. P. Orellana, M. B. Caminero, and C. Carrión, "On the provision of SaaS-level quality of service within heterogeneous private clouds," in *Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing (UCC '14)*, pp. 146–155, IEEE, London, UK, December 2014.

[2] J. Proaño, C. Carrión, and B. Caminero, "Towards a green, QoS-enabled heterogeneous cloud infrastructure," in *Proceedings of the 25th Heterogeneity in Computing Workshop in Conjunction with International Parallel and Distributed Processing Symposium (HCW-IPDPS '16)*, IEEE, Chicago, Ill, USA, May 2016.

[3] PANACEA: Proactive autonomic management of cloud resources, 2016, http://www.panacea-cloud.eu/.

[4] Microsoft Research: Cloud Security & Cryptography, April 2016, http://research.microsoft.com/en-us/projects/cryptocloud/.

[5] The Cloud Standards Wiki, http://cloud-standards.org/.

[6] R. Ranjan, B. Benatallah, S. Dustdar, and M. P. Papazoglou, "Cloud resource orchestration programming: overview, issues, and directions," *IEEE Internet Computing*, vol. 19, no. 5, pp. 46–56, 2015.

[7] Q. Zhang and W. Shi, "Energy-efficient workload placement in enterprise datacenters," *Computer*, vol. 49, no. 2, pp. 46–52, 2016.

[8] T. Guérout, T. Monteil, G. Da Costa, R. Neves Calheiros, R. Buyya, and M. Alexandru, "Energy-aware simulation with DVFS," *Simulation Modelling Practice and Theory*, vol. 39, pp. 76–91, 2013.

[9] Open Compute Project, http://www.opencompute.org/.

[10] R. Buyya, C. Vecchiola, and S. T. Selvi, *Mastering Cloud Computing: Foundations and Applications Programming*, Morgan Kaufmann, 1st edition, 2013.

[11] F. Chen, Y. Shan, Y. Zhang et al., "Enabling fpgas in the cloud," in *Proceedings of the 11th ACM International Conference on Computing Frontiers (CF '14)*, Cagliari, Italy, May 2014.

[12] T. Trader, *EU Projects Unite on Heterogeneous ARM-Based Exascale Prototype*, 2016, http://www.hpcwire.com/2016/02/24/eu-projects-unite-exascale-prototype.

[13] A. Putnam, A. M. Caulfield, E. S. Chung et al., "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceedings of the ACM/IEEE 41st International Symposium on Computer Architecture (ISCA '14)*, pp. 13–24, IEEE, Minneapolis, Minn, USA, June 2014.

[14] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency Computation Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[15] OpenCL, The Open Standard for Parallel Programming of Heterogeneous Systems, April 2016, http://www.khronos.org/opencl/.

[16] Vivado, Vivado Design Suite, April 2016, http://www.xilinx.com/products/design-tools/vivado/.

[17] J. Auerbach, D. F. Bacon, P. Cheng, and R. Rabbah, "Lime: a java-compatible and synthesizable language for heterogeneous architectures," in *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA '10)*, pp. 89–108, Reno, Nev, USA, 2010.

[18] M. Jacobsen and R. Kastner, "RIFFA 2.0: a reusable integration framework for FPGA accelerators," in *Proceedings of the 23rd International Conference on Field Programmable Logic and Applications (FPL '13)*, pp. 1–8, IEEE, Porto, Portugal, September 2013.

[19] K. Vipin and S. A. Fahmy, "DyRACT: a partial reconfiguration enabled accelerator and test platform," in *Proceedings of the 24th International Conference on Field Programmable Logic and Applications (FPL '14)*, pp. 1–7, IEEE, Munich, Germany, September 2014.

[20] C.-W. Tsai and J. J. P. C. Rodrigues, "Metaheuristic scheduling for cloud: a survey," *IEEE Systems Journal*, vol. 8, no. 1, pp. 279–291, 2014.

[21] S. Byma, J. G. Steffan, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "FPGAs in the cloud: booting virtualized hardware accelerators with OpenStack," in *Proceedings of the 22nd IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM '14)*, pp. 109–116, Boston, Mass, USA, May 2014.

[22] OpenStack, Open source software for building private and public clouds, April 2016, https://www.openstack.org/.

[23] S. A. Fahmy, K. Vipin, and S. Shreejith, "Virtualized FPGA accelerators for efficient cloud computing," in *Proceedings of the IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom '15)*, pp. 430–435, Vancouver, Canada, November 2015.

[24] Intel, Intel Virtualization Technology of Directed I/O, Architecture Specification, Rev. 2.2, 2014, http://www.intel.com/content/www/us/en/embedded/technology/virtualization/vt-directed-io-spec.html.

[25] D. Crockford, "The application/JSON media type for JavaScript object notation (JSON)," 2006.

[26] KVM, Kernel Based Virtual Machine (KVM), 2008, http://www.linux-kvm.org/.

[27] WeidongHan, How to assign devices with vt-d in kvm, 2009, http://www.linux-kvm.org/index.php?title=How_to_assign_devices_with_VT-d_in_KVM&action=info.

[28] R. M. Love, Taskset, 2004, http://linux.die.net/man/1/taskset.

[29] Xilinx ML605, Virtex-6 FPGA ML605 Evaluation Kit, March 2016, http://www.xilinx.com/publications/prod_mktg/ml605_product_brief.pdf.

[30] PowerMeterStore, Power Meter Store, March 2016, http://www.powermeterstore.com/p1206/watts_up_pro.php.

[31] R. da Rosa Righi, C. A. da Costa, V. F. Rodrigues, and G. Rostirolla, "Joint-analysis of performance and energy consumption when enabling cloud elasticity for synchronous HPC applications," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 5, pp. 1548–1571, 2016.