A Bi-objective Scheduling Approach for Energy Optimisation of Executing and Transmitting HPC Applications in Decentralised Multi-cloud Systems

Aeshah Alsughayyir and Thomas Erlebach
Department of Informatics
University of Leicester, UK
ayya1@mcs.le.ac.uk, t.erlebach@mcs.le.ac.uk

***Abstract*—Although cloud computing greatly utilises virtualised environments for applications to be executed efficiently in low-cost hosting, it has turned energy wasting and overconsumption issues into major concerns. Cloud infrastructure is built on a great amount of server equipment, including high performance computing (HPC), and the servers are naturally prone to failures.**

In this paper, we report on an energy optimisation approach for scheduling HPC applications, applied to decentralised clouds system, that takes dataset transmission energy into account. The optimisation supports combining two conflicting objectives: minimising energy consumption in conjunction with the avoidance of application deadline violations caused by resource failures. Furthermore, we propose two decision strategies for weighing these conflicting objectives dynamically to account for their significance towards producing an ideal energy efficiency and resource utilisation.

Through our developed simulation and experimental analysis using real parallel workloads from large-scale systems, the results illustrate that our approach provides promising energy savings with acceptable level of resource reliability.

***Index Terms*—online scheduling, non-preemptive tasks, DVFS advance reservation**

1. Introduction

The advent of cloud computing, although it greatly supports virtualised environments for applications to be executed efficiently in low-cost hosting, has turned energy wasting and overconsumption issues into major concerns. Cloud infrastructure is built upon a large number of servers, including high performance computing (HPC) and massive storage devices, that need huge energy supplies. In addition, it raises a concern regarding the reliability of resource utilisation, as cloud resources are naturally prone to failures [1].

A recent study [2] reports that cloud datacenters consume approximately up to 3% of the total energy consumption around the world, and the consumption is projected to grow significantly to 1012.02 billion kWh by 2020, a 35% growth [3]. This is due to the increasing need for cloud services in many sectors, such as the evolution of eScience and social big data analysis. A figure depicted in [4] also estimates that the majority of energy in a typical cloud datacenter is consumed by server computing activities, while less than 50% is needed for all other components such as storage and cooling systems.

In line with green cloud computing that recognizes the necessity of increasing energy efficiency and minimising global warming as well as air pollution [5], many pioneering researchers have devoted their studies, surveyed in [6], to improve the utilisation of cloud resources. To a large extent, the core factors behind efficient resource utilisation are the high capabilities of virtualisation techniques [7] as well as the flexibility of dynamically adjusting voltage and frequency of processors [8]. These techniques provide an effective way to save energy as virtualisation enables a reduction of the number of active physical machines (relying on Virtual Machine Migration and Consolidation techniques [6]) and, in turn, sharing bounded resources by virtually creating further machines or CPUs to potentially handle large workloads. Dynamic Voltage and Frequency Scaling (DVFS), however, enhances the energy efficiency of executing a workload.

Regarding the latter technique, i.e., DVFS, for supporting energy efficiency, the idea is to reduce the supplied voltage for a processor as much as possible while the desired performance, represented by an execution time bound, is still achievable [8]. In this setting, determining the critical frequency when scheduling a task depends mainly on measuring the status of resource utilisation. We have discussed in previous work [9] the determination of the best frequency for scheduling deadline-based tasks depending on the instantaneous status of the resources. The study focused on scheduling dependent HPC tasks on a decentralised multi-cloud system using best-effort (non-advance-reservation) mode, while in this paper we focus on the advance-reservation setting where the scheduling decision relies on precise data about resource availability.

To tackle energy-aware task scheduling over geographically distributed clouds, it is important to pay crucial attention to the energy consumed by dataset transmission. To our knowledge, this has not been addressed in the literature yet.

In addition to accounting for the energy consumption from both processing and dataset transmission while performing global scheduling, one faces a natural trade-off between energy minimisation and conflicting objectives such as quality-of-service optimisation.

In this paper, we propose energy optimisation algorithms for scheduling HPC applications, applied to decentralised cloud systems, taking the energy usage of dataset transmissions into account. The optimisation supports the combination of two conflicting objectives, minimising both energy consumption and application deadline violation caused by resource failures. The main contributions of this paper are:

- an energy-aware global scheduling algorithm with advance reservation (EGS) for allocating HPC applications to participating clouds, based on DVFS and cost of dataset transmission;
- an interdependent decision-making algorithm (referred to as combination rate strategy) to address conflicting objectives using a statistical approach;
- another decision-making algorithm (referred to as preference rate strategy) to optimise energy consumption based on setting an upper limit for the allowed energy consumption;
- an energy-aware local scheduling algorithm with advance reservation (ELS) for mapping each task to required resources.

The remainder of the paper is structured as follows. In Section 2 we give an overview of related work. Then, in Section 3, we introduce our system model and give the problem formulation. Section 4 discusses our proposed algorithms EGS and ELS. Section 5 describes the evaluation of our algorithms, before we conclude the paper with an outlook on future work in Section 6.

2. Related Work

The task scheduling problems taking into consideration energy-efficiency have been a hot subject of extensive research, see the approaches surveyed recently in [10], [6], [11]. The biggest difference between all the existing approaches and ours is that we consider (i) the energy cost of transferring datasets when globally scheduling applications over geographically distributed clouds and (ii) the occupation rate of cloud resources as a factor to minimise application deadline violations. The novelty here as far as this paper is concerned is to precisely schedule applications that consist of dependent tasks, based on a combination of resource occupation and two energy dimensions: energy consumed for execution and data transmission.

Apart from considering transmission energy, many approaches have been suggested to address the objective of minimising energy consumption from different perspectives such as datacenter management architecture [12], [13], scheduling workflows [14], reservation in mobile networks [15], or scheduling tasks in mixed-criticality systems [8]. In this section, we limit our discussion to closely relevant work [16], [17], [18] that focuses on the energy consumption problem for scheduling HPC applications in cloud computing systems.

Like this paper, the study in [16] also considers scheduling dependent tasks with deadline constraints for a HPC workflow and the conceptual use of scheduling levels: *global* and *local*. The *global* scheduling is applied to a distributed cloud system – their cloud system is centralised while ours is decentralised – to assign workflows to datacenters, whereas the *local* scheduling is for mapping tasks to machines. They use offline MultiObjective Evolutionary Algorithms (MOEAs) with the objective of optimising makespan, energy consumption, and deadline violations. Here, deadline violation can be accidentally caused by resource failures (e.g, when servers or network communications are down for maintenance) or by imprecise scheduling decisions due to the distributed environment. The latter is not expected to occur by our scheduler, since we adopt a token-based reservation schedule.

Wu et al. [17] propose a heuristic scheduling algorithm for heterogeneous computing environments, aiming to minimise power consumption without influencing the performance to satisfy a Service Level Agreement (SLA). In their approach, the minimum and maximum frequencies need to be specified with submitted tasks as well as SLA. For a given task, the scheduler repeatedly creates and assigns VMs from servers that remain within the required performance range until the task requirement is satisfied. If the requirement is still not satisfied, the scheduler turns on idle servers (if they exist) as required, allowing it to continue the creation and assignment process of VMs. Their method clearly has associated overhead costs when the scheduling fails. On top of this, it has been demonstrated by Juarez et al. [18] that creating or destroying VMs consumes non-trivial energy. Additionally, providing thresholds for the required frequency, in particular the maximum one, may limit the performance, which seems critical for deadline-based applications. Compared to their method, our approach differs in that our local scheduler ELS determines the best possible frequency to execute each task from the whole range of frequencies that is supported by the processors. We rely on both the provided computing volume per VM and the required number of machines by each task, while ensuring not to violate the deadline constraint of the whole application.

A scheduling approach for optimising a bi-objective function of either energy consumption or makespan in heterogeneous cloud systems was proposed by Juarez et al. [18]. They provide a combined cost function with a weighting factor α that indicates the user preference of either going for energy-efficiency or execution time. Their heuristic algorithm ranks tasks of a given Directed Acyclic Graph (DAG) by estimating the required energy. This is to determine independent subsets of tasks as a preparation step before allocating resources. In their method, the consumed energy is estimated by multiplying the task processing time by the proportional mean power. Compared to this, our energy model utilises DVFS where task execution time is multiplied by its instantaneous consumed energy that comprises

of both the static and the dynamic energy. The decision of our global scheduler relies on one of the proposed strategies: *preference rate* or *combination rate* strategy. The latter aims to minimise application deadline violations caused by resource failures alongside energy optimisation.

3. System Model and Problem Formulation

We consider a decentralised multi-cloud system that consists of a number of geographically distributed heterogeneous clouds, owned by different providers. They participate in a federated approach. The system consists of a set C of decentralised clouds, where $C = \{c_1, \cdots, c_k\}$, $k \in \mathbb{N}$. Each cloud c_j has a homogeneous datacenter, characterised by six parameters as described in Table 1. The manager server ms_j of each cloud relies on three components: a *global scheduler*, a *local scheduler*, and a *resource controller*. The latter acts as a resource checker, and is also responsible for query messages with participating clouds.

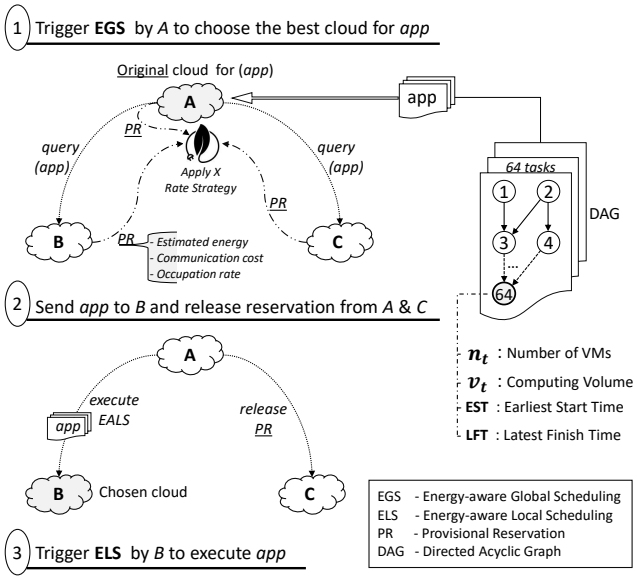

Figure 1. Overview of the approach, described in three steps

TABLE 1. CLOUD PARAMETERS OF A HOMOGENEOUS DATACENTER

Parameter	Description
ms_j	manager server
nS_j	number of servers
$nCPU_j$	number of physical processors per server
$capacity(s_{c_j})$	number of virtual machines per server
$[f_{min_j}, f_{max_j}]$	discrete frequency range of a processor
β_j and α_j	processor power parameters

Figure 1 gives an overview of our system model, illustrating the role of *global* and *local* schedulers when they receive a submitted application. Consider the cloud *A* that receives an application *app* with its specific requirements for execution. We call *A* 'original cloud' for *app*. The *global scheduler* in *A* sends queries to its local resource and also to all participating clouds (e.g., to *B* and *C*) by its *resource controller*. In response to such queries, the local resource checker of each cloud provides a provisional reservation *PR* that consists of estimated energy, communication cost and resource occupation rate, see *B* in Figure 1. These *PRs* are then analysed by the original cloud *A*, using the EGS algorithm, for deciding which cloud provides the best option for executing *app*. Assume *B* is the chosen cloud (see the bottom left of Figure 1). In this case, *A* will send messages to release *PR* from all unchosen clouds, and concurrently it sends the whole application *app* to the chosen cloud *B*. Here, the *local scheduler* of *B* applies a scheduling algorithm *ELS* for mapping tasks to machines, taking into account *app*'s requirements and its precedence constraints. In the remainder of this section, we divide the discussion of our system model into scheduling framework, energy models and problem formulation, referring to Figure 1 for illustration.

3.1. Scheduling Framework

Dynamic scheduling in an advance reservation model, using the token technique, depends principally on the actual available slots for a given period of future time. It needs to ensure that the time frame of an application is scheduled within the resource capacity, taking into account already occupied and reserved resources. Apart from the case of unexpected resource failures, this scheduling model is not expected to violate deadlines when executing an already scheduled application due to time or resource conflicts.

Before discussing our scheduling framework, we define the structure of applications followed in this paper. Consider the top right part of Figure 1, which illustrates the application as a directed acyclic graph (DAG). Here, an application $app_m = (V_m, E_m, ST_m, DL_m)$ consists of a set V_m of dependent tasks such that $V_m = \{t_1, \cdots, t_q\}$, $q \in \mathbb{N}$ and a set E_m of directed edges, each representing a data dependency between two tasks. The sets of direct predecessors and successors of a task t_i are denoted by $pred(t_i)$ and $succ(t_i)$, respectively. ST_m is the start time of app_m, and DL_m is the deadline. The original cloud of app_m is denoted by o_{app_m}.

A given task $t_i(n_{t_i}, v_{t_i}, EST_{t_i}, LFT_{t_i}) \in V_m$ is defined by four parameters as follows: n_{t_i} is the required number of VMs, v_{t_i} is the computing volume per VM, EST_{t_i} is the task's earliest start time, and LFT_{t_i} is the task's latest finish time. As a guide for the scheduler, EST_{t_i} and LFT_{t_i} of each task are calculated using (1) and (2), relying on start time ST_m and deadline DL_m of the submitted application.

$$EST_{t_i} = \begin{cases} ST_m & \text{if } t_i \text{ is entry} \\ \max_{t_m \in pred(t_i)}(EST_{t_m} + e_{t_m}) & \text{otherwise} \end{cases} \quad (1)$$

$$LFT_{t_i} = \begin{cases} DL_m & \text{if } t_i \text{ is exit} \\ \min_{t_m \in succ(t_i)}(LFT_{t_m} - e_{t_m}) & \text{otherwise} \end{cases} \quad (2)$$

Here, $e_{t_m} = \frac{v_{t_m}}{f}$ is the execution time of t_m at the minimum speed in a list of speeds that contains only the maximum speed of each cloud.

Our framework for scheduling submitted applications permits energy optimisation, in the first place, without affecting the desired performance. However, it is required that the deadline DL_m can be met by at least one participating cloud. In general, each submitted app_m can be either scheduled and then executed successfully, violated, i.e., scheduled but failing to get enough resources at execution time, or rejected.

The framework supports two scheduling strategies (preference-rate and combination-rate) for deciding the cloud that provides the best option, discussed in Section 4. Each cloud, including o_{app_m}, that meets DL_m should provide a provisional reservation (PR), consisting of:

i Estimated processing energy for all t_i.
ii Estimated data transmission energy for the input, the output and the disk image of app_m.
iii Resource occupation rate from ST_m to DL_m.

Given a set of PRs, the preference-rate strategy first selects a subset of this set by checking the maximum allowable energy of (i) and (ii) of each PR provided, then it gives priority to (iii). Combination-rate strategy, however, analyses (i) ,(ii) and (iii) of all provided PRs by dynamically adjusting the priority between estimated energy and occupation rate. Along with energy optimisation, it aims to avoid violation cases that may be caused by unexpected resource failures.

An application app_m is rejected if all clouds provide a negative PR. This means all clouds do not have enough resources to schedule app_m due to either their capacity limit or a tight DL_m.

3.2. Energy Model

We consider two aspects of the energy consumed by each app_m: execution activities and the dataset transmission between clouds (if o_{app_m} is not the executer one). Each of these aspects is discussed as follows.

3.2.1. Energy formula for execution activities. Our formula to compute execution energy, based on the processor details of a cloud site, is presented explicitly in (3). It calculates the total energy consumption E_{c_j} by a set of servers S in the cloud site c_j as follows:

$$E_{c_j} = \sum_{s \in S} \Big(\sum_{p \in P(s)} \big((\beta_j + \sum_{co \in CO_p} \alpha_j f_{co}^3) D_p \big) \Big) \quad (3)$$

where $P(s)$ is the set of processors of server s, D_p denotes the active time of processor p, and f_{co} denotes the frequency level at which core $co \in CO_p$ for some processor p runs.

The formula (3) accounts for the energy consumption by executing tasks or even only by the processor being active. In particular, it is affected by the static energy use besides the execution activities, i.e., dynamic energy and leakage current. The processor energy consumption can be controlled by employing the dynamic voltage and frequency scaling DVFS technique. It allows adjusting the processor frequency up or down in order to manage its dynamic energy and enhance energy saving as much as possible. The lower the processor frequency is, the less instantaneous energy it consumes, but incurring a longer execution time.

However, due to the convexity of the energy metric (3), not all lower frequency levels are useful for minimising the energy consumption. We can define a useless frequency f to be a frequency at which the processor, when executing a fixed volume of computation, always dissipates an amount of energy that is larger than the amount of energy dissipated at the frequency m that minimises the amount of energy, and if f belongs to the interval $[f_{min}, m)$. Figure 2 shows an example of the energy consumption per unit of computation of a dual-core processor with $\alpha = \beta = 60$ and a number of discrete frequency levels in the range $[0.17, 2.5]$, where the interval of the useless frequencies is $[0.17, m)$. Despite the fact that these useless frequencies may have an instantaneous energy consumption that is lower than that of higher frequencies, they always need more energy in total for executing a task than some higher frequency that finishes the task sooner.

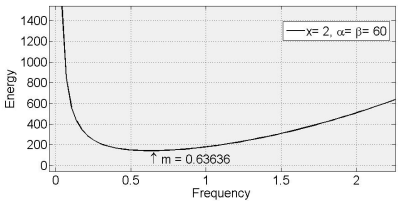

Figure 2. Energy consumption vs. frequency

Thus, we eliminate useless frequencies as follows. For each frequency f we compute the amount of energy consumption of a processor for one unit of computation on each core, which is given by $Energy = \frac{\beta + (x * \alpha f^3)}{f}$ for a processor with x cores. We then remove all frequencies that are smaller than the frequency that minimises this energy. This elimination, where applicable, helps to reduce the computation time of the scheduling method that selects a suitable frequency for executing a task.

Moreover, we calculate the estimated energy consumption by a particular task t as follows. Assume that n_t is the number of VMs assigned to t such that each VM occupies one core, and these VMs run at frequency f. Then, the total energy consumption by task t can be expressed as:

$$E_t = (\beta + N(\alpha f^3)) \times ceil(\frac{n_t}{N}) \times e_t \quad (4)$$

where $e_t = \frac{v_t}{f}$ is the execution time of task t at frequency f, N is the number of cores per processor (assuming also a single VM per core), and $ceil(\frac{n_t}{N})$ represents the estimated number of physical processors that are assigned to t.

3.2.2. Energy formula for dataset transmission. The computation of transmission energy depends mainly on the cost of wired connections through which the dataset of a given size is transmitted. The link cost combines the total consumption of the Internet nodes and cooling, the transmission lines and amplifiers [19]. When a non-original cloud (i.e., not o_{app_m}) executes an application app_m, the datasets, including the input and the disk image, need to be transmitted to the executer cloud. Once the execution of app_m is completed, the output will also need to be transmitted back to the o_{app_m}.

Estimating the energy consumption of data transmissions through the Internet is notoriously difficult, and available estimates vary by several orders of magnitude [19], [20]. We adopt an estimate of $0.2\ kWh$ for the transmission of 1 GB as this value lies in the middle region of the range of reported estimates. Furthermore, to account for the effect that transmissions over longer distances are likely to require more hops and thus more energy, we assume that the energy consumption of a data transmission also depends linearly on the distance over which the data is being transmitted. We make the assumption that a typical transmission to which the rate of $\mu = 0.2kWh/GB$ applies is a national transmission over a distance of $500\ km$, so that the energy cost of a transmission over a distance D can be obtained by multiplication with the factor $D/500\,km$. (If a more accurate estimation of transmission energy costs is available for a given scenario, it can be incorporated into our scheduling algorithms in a straightforward way.)

We assume that all datasets of an application will be sent through the same link. Given a set of delegated applications whose datasets need to be sent from o_{app_m} to the executor cloud, we estimate transmission energy T_{c_j} as:

$$T_{c_j} = \sum_{app_m \in A} (\mu \times dataSize_{app_m} \times \frac{linkLG_{app_m}}{500\ km}) \qquad (5)$$

where $dataSize_{app_m}$ denotes the total size of the disk image, the input and output, and $linkLG_{app_m}$ expresses the link-length used for the transmission.

3.3. Problem Formulation

We consider a set of applications $\mathcal{A}$, submitted over time to different specified cloud sites in a multi-cloud system, where $\mathcal{A} = \{app_1, \ldots, app_L\}, L \in \mathbb{N}$. Cloud c_j may receive y applications, where $0 \leq y \leq L$. The submission of app_m is unknown beforehand. Each cloud can accept a received app_m if the deadline can be met, or reject it otherwise. If the app_m gets accepted for scheduling, it will be either executed successfully or violated. Our objective is to primarily optimise the total energy consumption of all accepted applications in the entire multi-cloud system with the avoidance of rejections and application violation cases.

We attempt to minimise the overall energy usage E_{total} that includes (1) the computing energy usage E_{c_j}, (2) the dataset transmission energy cost T_{c_j} and (3) the penalty cost for rejecting/violating applications PN_{c_j} by cloud c_j. Here, the penalty cost PN for rejecting/violating an application app_m is $PN = \sum_{t \in app_m} E_t$, where E_t is calculated by equation (4) at the highest performance among all clouds. Thus, the objective function is as follows.

Minimise: $E_{total} = \sum_{c_j \in C}(E_{c_j} + T_{c_j} + PN_{c_j})$
Subject to:

(1) $endTime_{app_m} \leq DL_m \quad \forall\ app_m \in \mathcal{A}$ where app_m is accepted

(2) $f_{min_j} \leq f_{co} \leq f_{max_{c_j}} \quad \forall\ co$ in servers of $c_j \in C$

By this objective function, a scheduling policy that rejects all applications would have $\sum_{c_j \in C} E_{c_j} = \sum_{c_j \in C} T_{c_j} = 0$, but it gets a very high $\sum_{c_j \in C} PN_{c_j}$. The policy that executes all applications at the highest cloud performance would tend to have a very low penalty but a high $\sum_{c_j \in C} E_{c_j}$. The scheduling policy which will have a better objective value is the one that finds a good balance.

4. Global and Local Scheduling Algorithms

This paper makes original contributions to optimising energy consumption when scheduling HPC applications over distributed multi-cloud systems in two aspects. First of all, it proposes *EGS*, an advance token-reservation based algorithm as a novel global scheduling for allocating a submitted application to the best cloud in the system. *EGS* supports minimising application violation cases based on the proposed CRS strategy. It considers gathering two energy costs when globally scheduling an application, which are (i) execution energy at the CPU level and (ii) dataset transmission (if applicable) at the network level. Second of all, *ELS* focuses on scheduling application tasks locally in cloud resources using the dynamic voltage and frequency scaling technique. The designed algorithms are presented in detail in the remainder of this section.

4.1. EGS: Energy-aware Global Scheduling with Advance Reservation

As described in Figure 1, EGS takes as input a submitted application and a set of provisional reservations PRs from participating clouds to pick the best PR provided. It is triggered by o_{app_m} upon the arrival of all the positive responses from clouds, i.e., offering the ability of scheduling the submitted application with meeting its deadline. Each PR consists of both $eEnergyValue$ and $occupRt$ and has a limited token period of validity starting from the time of response, which enables a cloud provider to release its resources by cancelling its PR if no confirmation is received from o_{app_m} within the allowed time.

EGS is performed based on one of the two proposed strategies: PRS or CRS (cf. Section 3.1), explained as follows:

- *Preference Rate Strategy (PRS).*
  PRS minimises primarily the energy consumption based on a given preference factor Rt that can be chosen as any fraction in the range [0,1). The idea is to determine the acceptable energy costs by assigning the $min(eEnergyValue)$ provided as the lower bound, while choosing $min(eEnergyValue) + (min(eEnergyValue) \times Rt)$ as the upper bound. Only PRs whose $eEnergyValue$ lies in this range are then considered, and the strategy then minimises $occupRt$ among those PRs.
  Intuitively, if the given factor $Rt == 0$, *EGS* would always choose the cloud that gives the minimum energy immediately without considering $occupRt$. Accordingly,

the applications will be executed at the minimum offered energy consumption.

- *Combination Rate Strategy (CRS).*
  Unlike PRS, CRS aims to simultaneously satisfy the minimisation of both the estimated $eEnergyValue$ and $occupRt$. The strategy is inspired by two statistical analysis concepts that are the standard deviation SD and the coefficient of variation RSD. The SD for a set of $eEnergyValue$ expresses how much the proposed energy values differ from their mean value, e.g., for a set $Energy = \{en_1, \cdots, en_a\}$, SD can be calculated by (6):

$$SD = \sqrt[2]{\frac{\sum_{en_i \in Energy} |en_i - \overline{Energy}|^2}{a}} \tag{6}$$

  where $\overline{Energy}$ is the mean of the data set $Energy$, calculated by $\overline{Energy} = \frac{\sum_{en \in Energy} en}{a}$. The coefficient of variation RSD is the ratio of standard deviation to the mean $\overline{Energy}$, and can be expressed as: $RSD = SD/\overline{Energy}$.
  To pick the best cloud based on this strategy, the EGS forms two lists $eEnergyList$ and $occRtList$ with all proposed $eEnergyValue$ and $occupRt$. Here, the RSD of each list represents the amount of dispersion between the elements such that low dispersion would refer to very similar proposed values, which may make no difference when choosing any element. High dispersion, however, means that it is important to consider each element as there is a clear difference between the elements.
  Having the RSD from $eEnergyList$ and $occRtList$, it makes sense to use these to determine a weight for each objective in a weighted sum. More precisely, the higher weight hw will be given to the set of items that are highly dispersing and the lower weight lw to the other list. Assume that $eEnergyValue$ has a higher weight hw, the *EGS* will choose the cloud with minimum combined rate from the list: $\{(eEnergyValue_1 * hw + occRtList_1 * lw), \cdots, (eEnergyValue_k * hw + occRtList_k * lw)\}$, where k is the number of received provisional reservations.

The pseudo code presented in Algorithm 1 and 2 gives a high-level view of our EGS algorithm. A participating cloud that is able to schedule an application will provide a PR, consisting of a positive estimation of $eEnergyValue$ for processing and transmitting the dataset, see *lines 1-5* of Algorithm 1. A negative $eEnergyValue$, however, means that the provider cloud cannot satisfy the application's deadline, and thus the return pr will not be added in the list PRs. In *lines 6-9*, the algorithm determines the output of either rejecting app if none of the clouds can schedule it (i.e., $PRs == \emptyset$), or selecting a cloud immediately if only one positive option is found (i.e., $PRs.size == 1$).

If more than one cloud can execute the app, the decision, based on either PRS or CRS, of which cloud will execute it is described in *lines 11-12* of Algorithm 1. In *lines 13-14*, all unchosen clouds are notified to release their pr.

Algorithm 1 EGS
Energy-aware global scheduling with advance reservation.

Inputs: app is a submitted application to o_{app}
$C = \{c_1, \cdots, c_k\}$, $k \in \mathbb{N}$ is a set of connected clouds
PRS (preference) and CRS (combination) are rate strategies.
Outputs: A globally scheduled app in c_i the best cloud found.
Begin
1: $PRs := \emptyset$ is a set of provisional reservations provided from all c_i
2: **for each** ms_i cloud manager server $\in C$ **do**
3: $pr \leftarrow ms_i$.makeReservation(app) expanded in Algorithm 2
4: **if** $pr.eEnergyValue > 0$ **and** $pr.Token.isValid$ **then**
5: $PRs \leftarrow pr$
6: **if** $PRs == \emptyset$ **then**
7: **set** app.isRejected=true
8: **else if** $PRs.size == 1$ **then**
9: **set** chosenCloud($app, PRs[1]$)
10: **else**
11: set $pr \leftarrow$ applyDecisionStrategy (PRs, CRS (or PRS))
12: chosenCloud (app, pr)
13: removeFrom (PRs, pr)
14: releasePR (app, PRs)
End

Algorithm 2 makeReservation

Inputs: $app_m(V_m, E_m, ST_m, DL_m)$.
Outputs: a token that is associated with PR.
Begin
1: $CPUcap :=$ is the processors capacity of this cloud.
2: **for** $i := 1$ to q **do**
3: $Thr := CPUcap - n_{t_i}$
4: $P :=$ is the number of used processors at EST_{t_i}.
5: $bestIntvl := \emptyset$
6: **if** $Thr < 0.0$ **then**
7: return $token := 0$.
8: **if** $Thr \geq P$ **then**
9: $inPeak := false$ **and** $startAv := EST_{t_i}$.
10: **else**
11: $inPeak := true$.
12: get all overlapping tasks and form list of start and end points $PT = \{pt_1, \cdots, pt_b\}$, excluding any start point pt where $pt \leq EST_{t_i}$.
13: **for** $j := 1$ to b **do**
14: **if** pt_j is a start point **then**
15: $P = P + n_{pt_j}$
16: **if** $inPeak = false$ and ($P > Thr$ or $pt_j = LFT_{t_i}$) **then**
17: try to schedule t_i in $[startAv, pt_j]$ and get its f
18: **if** f is the minimum frequency in this cloud **then**
19: $bestIntvl := [startAv, pt_j]$ *then* break
20: **else if** $[startAv, pt_j]$ is longer than $bestIntvl$ **then**
21: $bestIntvl := [startAv, pt_j]$
22: $inPeak := true$
23: **else**
24: **if** pt_j is an end point **then**
25: $P = P - n_{pt_j}$.
26: **if** $inPeak = true$ and $P \leq Thr$ **then**
27: $inPeak := false$ **and** $startAv := pt_j$.
28: **if** $bestIntvl \neq \emptyset$ **then**
29: schedule t_i to $bestIntvl$ and update the EST of its $succ(t_i)$
30: calculate the estimated energy consumption E_{t_i}.
31: $eEnergyValue = eEnergyValue + E_{t_i}$
32: **else**
33: return $token := 0$.
34: calculate $occupRt$.
35: generate unique $token$ and associate it with the scheduling.
36: return $token$.
End

4.2. ELS: Energy-aware Local Scheduling Algorithm

For each task t_i in the application app, the ELS algorithm is responsible for assigning t_i to servers, processors, and cores that will execute it, according to the schedule by Algorithm 2. The ELS is triggered whenever the time for executing a t_i is due to start as it allocates the required machines to the task for execution. The app will be violated if t_i has failed to execute (e.g., by not getting enough resources at run time), which means the schedule of all its $succ(t_i)$ will be cancelled.

The goal of Algorithm 3 is to choose the resources in a way that helps minimising the computing energy consumption. Thus, it initially tries to utilise as many active servers as possible, in *line 4*, so as to reduce the cost of activating ideal servers. It also utilises the active processors that have free virtual capacity in order to minimise the static energy consumption, see *lines 5-9*.

Algorithm 3 ELS
Energy-aware local scheduling with advance reservation.

Inputs: Task $t_i(n_{t_i})$.
$capacity(S_c)$ the capacity of servers in this cloud.
The list of all servers.
Outputs: Allocating the required machines to t_i.
Begin
1: $Rn := n_{t_i}$
2: form the list of all active servers $activeServersList$
3: sort the $activeServersList$ in ascending order of their free capacity.
4: **for each** server $s \in activeServersList$ **do**
5: form the list of processors that have free capacity $CPUsList$
6: sort the $CPUsList$ in ascending order of their free virtual capacity.
7: **for each** processor $p \in CPUsList$ **do**
8: allocate a number of VMs that fulfill Rn if available, or equal to the number of free VMs otherwise.
9: reduce Rn value by the number of allocated VMs.
10: **if** $Rn = 0$ **then**
11: break
12: **if** $Rn > 0$ **then**
13: activate $ceil(Rn/capacity(S_c))$ idle servers.
14: allocate the remaining number of required VMs that is equal to Rn.
15: start executing the task t_i.
End

5. Experimental Evaluation

This section presents the experiments conducted to evaluate the proposed schedulers, concentrating on two aspects:

- Measuring the effect of the proposed schedulers on the energy saving based on the general objective function cost (i.e., the total of energy usage plus penalty for rejected/violated applications). It gives the average reduction with respect to different application workloads.
- Measuring the impact of resource failures (i.e., failures that may occur accidentally in the cloud system) on already scheduled applications. This is to get a rough idea of how our proposal can contribute to providing a reliable scheduler.

Finally, this section will discuss the effectiveness of the proposed strategies, PRS and CRS, for different workloads of applications with respect to (i) the average reduction of energy and (ii) the number of HPC application rejections and violations.

5.1. Configurations

We have used an improved version of our simulation tool, presented in [9], mainly extended to handle the resource reservation technique. The simulation experiments use a decentralised multi-cloud system of five distributed clouds around the world. The characteristics of these clouds including approximate distances between them are shown in Table 2 and Table 3. In each cloud site, we assume the capacity of VMs per server is twice the number of its physical processors, and all the processors support 5 levels of frequency in $[f_{min}, f_{max}]$, where f_{min} is 37.5% of f_{max}.

TABLE 2. SPECIFICATION OF THE FIVE CLOUDS USED IN OUR SIMULATION.

Data center	Total #VMs	Performance (TFLOP/s)	CPU parameters		
			α	β	f_{max}
WestUSA	32000	0.0072	7.5	65	1.8
Germany	32000	0.0096	60	60	2.4
Japan	32000	0.012	4.5	90	3.0
India	32000	0.0128	4.0	90	3.2
Brazil	32000	0.0128	4.4	105	3.2

TABLE 3. APPROXIMATE DISTANCES IN KM BETWEEN THE CLOUD DATACENTERS.

—	WestUSA	Germany	Japan	India	Brazil
W	—	9094.4	8632.4	13365.2	9058.6
G	9094.4	—	9058.5	6759.7	9442.2
J	8632.4	9058.5	—	5965.9	17389.8
I	13365.2	6759.7	5965.9	—	16201.9
B	9058.6	9442.2	17389.8	16201.9	—

TABLE 4. THREE CATEGORIES OF PARALLEL WORKLOAD APPLICATIONS.

Category	Max. n_{t_i}	# applications	# tasks in each app
Low-load	8696	200	64
Mid-load	11384	200	64
High-load	16384	200	64

Table 4 describes three categories of parallel application workloads extracted from different logs of real large-scale systems (i.e., LLNL-uBGL-2006-0, LLNL-Thunder-2006-0, LLNL-Atlas-2006-0, and ANL-Intrepid-2009-1) [21], [22]. The task dependencies are inferred, as in [23], from the provided start and finish times of executing jobs in each log.

We assume the deadline to execute a submitted app_m is calculated by its estimated execution time $eExecTime_{app_m}$, extended by 20% in the case of loose deadline, and by 0.1% for tight deadline. For example, the loose deadline DL_m is calculated as $DL_m = eExecTime_{app_m} + (eExecTime_{app_m} \times 0.2)$. Moreover, all tasks of app_m are

CPU-bound. The applications are submitted to the multi-cloud system at different times, and the gap interval between each two consecutive application submissions is equal to 1000 seconds.

5.2. Experimental Results

To evaluate the EGS and ELS algorithms with both PRS and CRS strategies, the highest frequency mode is considered as an upper bound of energy usage. It models the case when the objective of the cloud providers is to offer their services at the highest performance. It is also applied with PRS (referred to as PRS.HF mode) and CRS (referred to as CRS.HF mode) to attempt minimising energy usage. We assume that Rt is equal to 0.01 for the PRS strategy to allow choosing the cloud that gives the minimum energy immediately without considering $occupRt$.

In addition to the general objective function as a metric, the rate of total energy usage is calculated to compare the different scheduling strategies by $\frac{\sum_{c_j \in C}(E_{c_j}+T_{c_j})}{\sum_{app_m \in \mathcal{A}'} \sum_{t \in app_m} v_t}$. Here, E_{c_j} and T_{c_j} are the amount of energy usage by cloud c_j for execution and transmission activities, $\mathcal{A}'$ is the set of applications that are successfully completed, and v_t is the computing volume of the executed task t that belongs to a successfully completed application app_m.

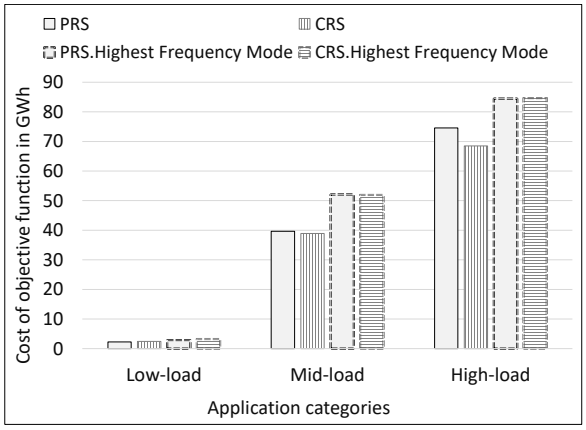

Figure 3. Different scheduling strategies with various workload categories vs. the objective function.

5.2.1. Effect of the proposed schedulers on energy optimisation. Figure 3 shows the total energy cost achieved by PRS, CRS, PRS.HF and CRS.HF according to the proposed objective function that is applied on the various workloads. On the one hand, it is clear from the chart that PRS and CRS produce a lower energy cost than PRS.HF and CRS.HF in all cases by an average of about 19.3% and 23.1%, respectively. On the other hand, the chart shows a considerable difference in the total energy cost of PRS and CRS, with PRS being smaller than CRS by about 7.5% in the low-load case. However, CRS produces total energy cost less than the one by PRS by about 1.97% with mid-load and about 8.15% with high-load. This indicates that whenever the workload gets heavier the CRS strategy produces better results in terms of total energy cost.

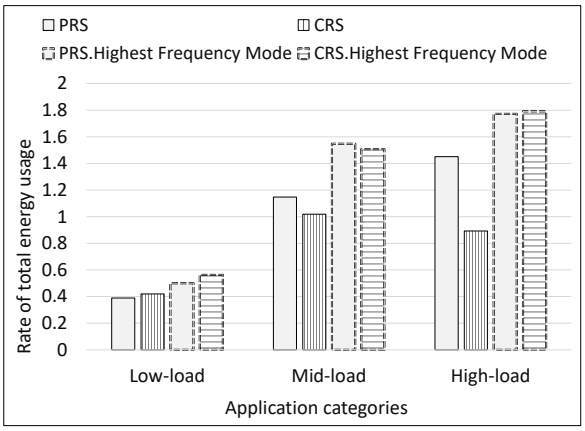

Figure 4. Different scheduling strategies with various workload categories vs. the rate of energy usage.

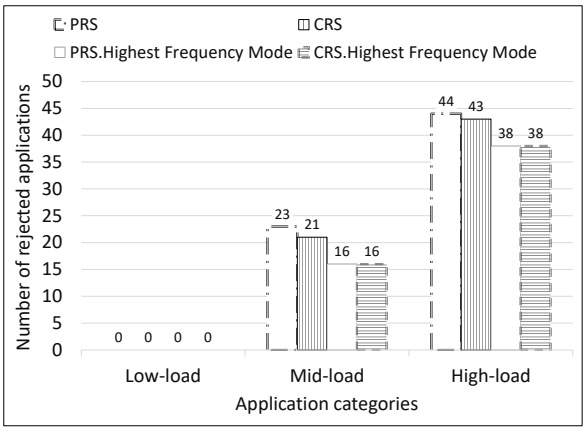

Figure 5. Different scheduling strategies with various workload categories vs. the number of rejected applications.

Considering the rate of the energy consumed by executing tasks and transmitting datasets, shown in Figure 4, the difference between PRS, CRS, PRS.HF, and CRS.HF is still evident with the various workload categories. Despite the fact that CRS computed more volume than PRS due to a lower number of rejected applications with both mid-load and high-load, as shown in Figure 5, it has a lower rate of energy usage than PRS by an average of about 24.8%. Moreover, PRS.HF and CRS.HF produced different rates of energy usage, although they rejected the same number of applications with the different workload categories.

5.2.2. Impact of the resource failures on scheduled applications. Regarding the level of resource reliability while scheduling, we simulate ELS in scenarios where a percentage of servers (growing in increments of 6%) become

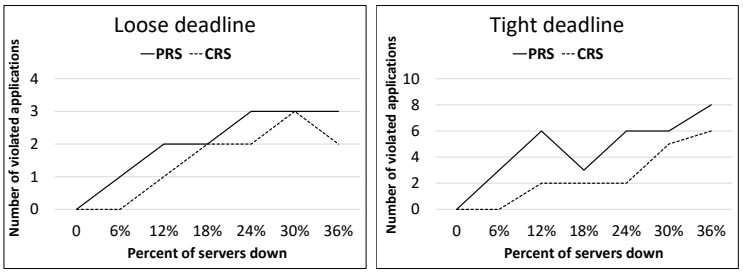

Figure 6. Number of violations vs. percentage of down servers

unavailable at runtime. For each execution, the failures are triggered from time 0, and stay in a failed state until the end of the simulation. Figure 6 shows the number of violations occurring due to the increased number of unavailable servers in all cloud sites for inputs with loose and tight deadlines. The experiments are performed on 40 submitted applications that are randomly mixed from the low-load and mid-load categories. In the case of loose deadlines CRS achieved a number of violations lower than PRS by an average of 36.1%, while it was about 51.4% lower in the tight deadlines case. This reflects the positive effect of the dynamic consideration of resource rate occupation by CRS compared to PRS.

5.3. Discussion

The experimental results illustrate that scheduling HPC applications, focusing on optimising overall energy consumption, is affected by several interdependent factors. The affecting elements we want to shed light on are the kinds of applications in terms of their requirements and the status of resource occupation when an application is received for scheduling.

In general, scheduling an application to a cloud that appears better at the submission time may not lead to the best energy saving result over time. Specifically, in the case of high-load applications, CRS produces better energy savings than PRS due to the dynamic technique of balancing the workloads among all clouds when applicable.

Figure 7 and Figure 8 describe the performance of all clouds in the system when submitting 200 high-load applications by running PRS and CRS, respectively. Here, we can observe the behaviours of resource occupation in the multi-cloud system. PRS always exploits heavily the cloud that gives the lowest energy (e.g., see *Japan* for the submissions 1 - 20), causing in some cases the applications scheduled later to be allocated to less efficient clouds. As CRS tries to balance the level of resource utilisation over all clouds, it chose firstly *Germany* which has a higher α (i.e., seems to consume higher energy than *Japan*), then *India*, see Figure 8. Considering the overall reduction of energy, CRS gives better results than PRS for medium to high load applications. We can list the main findings of our study as follows:

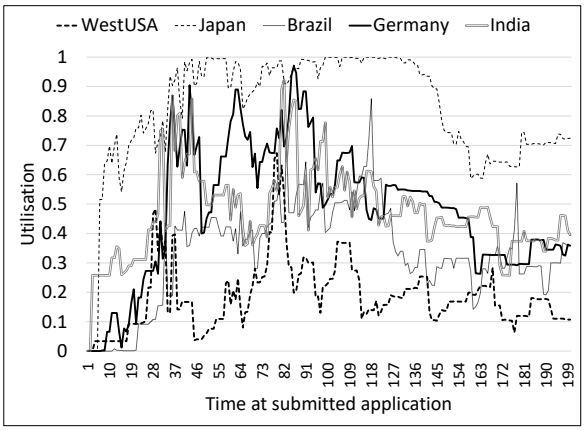

Figure 7. Resource utilisation with preference rate Strategy PRS

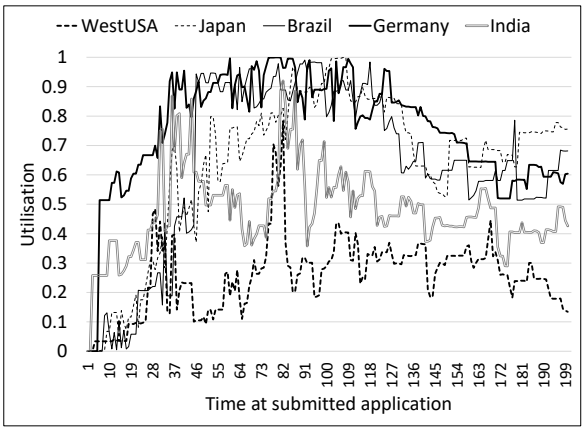

Figure 8. Resource utilisation with combination rate strategy CRS

- The best energy cost saving of about 23%, based on our objective function, is obtained by CRS compared to its upper bound CRS.HF.
- None of the strategies (i.e., PRS or CRS) proves itself to be the best with any submitted workloads in terms of energy efficiency, where PRS shows better results with low-load compared to CRS.
- Application deadline violations can be avoided for both tight and loose deadlines with CRS being better than PRS by an average of 43%.
- As we use token-based reservation, the token validity time and the arrival-gap of submitted applications are crucial factors impacting on the number of application rejections.

6. Conclusion

Approaches to the scheduling of HPC applications with the goal of energy optimisation should not focus on just the single parameter of energy consumption but incorporate different parameters, ranging from CPU usage levels to

data transmissions at network level. In multi-cloud systems, optimal scheduling for energy efficiency that relies on resource utilisation needs to pay special attention to resource reliability. This paper has focused on combining two different aspects of energy usage while scheduling HPC applications and has considered simultaneously minimising application rejections and deadline violations, to support resource reliability, with energy optimisation.

The conducted experiments using our simulation have shown that our scheduling approach using the proposed strategies can reduce energy consumption by an average of 21% as compared to the upper bound, determined by the highest performance possible in the cloud-system. The results have revealed that a significant issue in energy aware scheduling is that a designed mechanism that depends only on the absolute minimum energy value to execute an application may not necessarily produce the best overall energy saving in all cases.

Furthermore, the results have shown that there is an interdependency between using one of the proposed strategies for scheduling decisions and the characteristics of the submitted applications. Consequently, we plan to further optimise energy consumption by designing an adaptive algorithm that can dynamically adjust the decision strategy (if needed) based on the given scheduling problem.

We will be investigating on designing a rescheduling mechanism, relying mainly on the release of resource reservation, that may help in optimising further energy consumption as well as application rejection cases.

Acknowledgment

The first author would like to thank the department of computer science in Taibah University in Medina for partially supporting this work.

References

[1] Y. Sharma, B. Javadi, W. Si, and D. Sun, "Reliability and energy efficiency in cloud computing systems: Survey and taxonomy," *J. Network and Computer Applications*, vol. 74, pp. 66–85, 2016.

[2] "Gartner's strategic technology trends for 2017," 2016. [Online]. Available: http://www.gartner.com/technology/home.jsp

[3] J. Whitney and P. Delforge, "Scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers. issue paper," *Natural Resources Defense Council (NRDC)*, 2014.

[4] X. Wang and Y. Wang, "Energy-efficient multi-task scheduling based on mapreduce for cloud computing," in *Seventh International Conference on Computational Intelligence and Security, CIS 2011, Sanya, Hainan, China, December 3-4, 2011*, 2011, pp. 57–62.

[5] R. Buyya, A. Beloglazov, and J. H. Abawajy, "Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges," *CoRR*, vol. abs/1006.0308, 2010.

[6] T. Kaur and I. Chana, "Energy efficiency techniques in cloud computing: A survey and taxonomy," *ACM Comput. Surv.*, vol. 48, no. 2, pp. 22:1–22:46, 2015.

[7] I. Rodero, J. Jaramillo, A. Quiroz, M. Parashar, F. Guim, and S. Poole, "Energy-efficient application-aware online provisioning for virtualized clouds and data centers," in *International Green Computing Conference 2010, Chicago, IL, USA, 15-18 August 2010*, 2010, pp. 31–45.

[8] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Energy efficient dvfs scheduling for mixed-criticality systems," in *Proceedings of the 14th International Conference on Embedded Software*, ser. EMSOFT '14. New York, NY, USA: ACM, 2014, pp. 11:1–11:10.

[9] A. Alsughayyir and T. Erlebach, "Energy aware scheduling of HPC tasks in decentralised cloud systems," in *24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2016, Heraklion, Crete, Greece, February 17-19, 2016*, 2016, pp. 617–621.

[10] I. Pietri and R. Sakellariou, "Mapping virtual machines onto physical machines in cloud computing: A survey," *ACM Comput. Surv.*, vol. 49, no. 3, pp. 49:1–49:30, Oct. 2016.

[11] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, "Cloud computing: Survey on energy efficiency," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 33:1–33:36, Dec. 2014.

[12] J. Guitart, "Toward sustainable data centers: a comprehensive energy management strategy," *Computing*, pp. 1–19, 2016.

[13] A.-C. Orgerie and L. Lefèvre, "Energy-efficient reservation infrastructure for grids, clouds and networks," *Energy Efficient Distributed Computing Systems*, pp. 133–162, 2012.

[14] S. Iturriaga, B. Dorronsoro, and S. Nesmachnow, "Multiobjective evolutionary algorithms for energy and service level scheduling in a federation of distributed datacenters," *International Transactions in Operational Research*, vol. 24, no. 1-2, pp. 199–228, 2017.

[15] E. J. Vergara, S. Nadjm-Tehrani, and M. Asplund, "Sharing the cost of lunch: Energy apportionment policies," in *Proceedings of the 11th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, ser. Q2SWinet '15. New York, NY, USA: ACM, 2015, pp. 91–97.

[16] S. Iturriaga, S. Nesmachnow, A. Tchernykh, and B. Dorronsoro, "Multiobjective workflow scheduling in a federation of heterogeneous green-powered data centers," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2016, pp. 596–599.

[17] C. Wu, R. Chang, and H. Chan, "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters," *Future Generation Comp. Syst.*, vol. 37, pp. 141–147, 2014.

[18] F. Juarez, J. Ejarque, and R. M. Badia, "Dynamic energy-aware scheduling for parallel task-based application in cloud computing," *Future Generation Computer Systems*, 2016.

[19] V. C. Coroama and L. M. Hilty, "Assessing internet energy intensity: A review of methods and results," *Environmental impact assessment review*, vol. 45, pp. 63–68, 2014.

[20] V. C. Coroama, D. Schien, C. Preist, and L. M. Hilty, "The energy intensity of the internet: home and access networks," in *ICT Innovations for Sustainability*. Springer, 2015, pp. 137–155.

[21] "Parallel Workloads Archive," http://www.cs.huji.ac.il/labs/parallel/workload/index.html, 2005, [Online; accessed 14-May-2017].

[22] D. G. Feitelson, D. Tsafrir, and D. Krakov, "Experience with using the parallel workloads archive," *J. Parallel Distrib. Comput.*, vol. 74, no. 10, pp. 2967–2982, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2014.06.013

[23] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on IaaS cloud systems," *Journal of Parallel and Distributed Computing*, vol. 72, no. 5, pp. 666 – 677, 2012.