Future Generation Computer Systems 96 (2019) 216–226

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations

Georgios L. Stavrinides [*], Helen D. Karatza

Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece

HIGHLIGHTS

- The proposed scheduling heuristic consistently outperformed the baseline policies.
- Only an insignificant loss of job result precision was incurred.
- In the worst case, the provided job results were only about 0.82% imprecise.

ARTICLE INFO

Article history:
Received 4 November 2018
Received in revised form 22 January 2019
Accepted 11 February 2019
Available online 15 February 2019

Keywords:
Quality of Service
Energy efficiency
Real-time workflows
Per-core DVFS
Approximate computations
Scheduling

ABSTRACT

Green cloud computing attracts significant attention from both academia and industry. One of the major challenges involved, is to provide a high level of Quality of Service (QoS) in a cost-effective way for the end users and in an energy-efficient manner for the cloud providers. Towards this direction, this paper presents an energy-efficient, QoS-aware and cost-effective scheduling strategy for real-time workflow applications in cloud computing systems. The proposed approach utilizes per-core Dynamic Voltage and Frequency Scaling (DVFS) on the underlying heterogeneous multi-core processors, as well as approximate computations, in order to fill in schedule gaps. At the same time, it takes into account the effects of input error on the processing time of the component tasks. Our goal is to provide timeliness and energy efficiency by trading off result precision, while keeping the result quality of the completed jobs at an acceptable standard and the monetary cost required for the execution of the jobs at a reasonable level. The proposed scheduling heuristic is compared to two other baseline policies, under the impact of various QoS requirements. The simulation experiments reveal that our approach outperforms the other examined policies, providing promising results.

1. Introduction

As cloud computing continues to gain momentum, providing a high level of Quality of Service (QoS) in a cost-effective way for the end users and in an energy-efficient manner for the cloud providers, is an ongoing challenge that gathers significant attention from both academia and industry. A Service Level Agreement (SLA) is commonly required between the end users and a cloud provider. It is a contract between the two parties that defines the type of the provided service, the QoS requirements, as well as the adopted pricing scheme [1].

Compared to other data center infrastructure components, processors typically consume the greatest amount of energy [2–4]. A widely used power management method is the *Dynamic Voltage and Frequency Scaling (DVFS)* technique. DVFS allows the dynamic adjustment of the supply voltage and operating frequency (i.e., speed) of a processor, based on the workload conditions [5,6]. Modern multi-core processor architectures incorporate voltage regulators for each integrated core, allowing per-core DVFS, so that each core can operate at a different voltage and frequency level from the other cores of the same processor [7]. While this provides flexibility and better energy efficiency, it involves higher control complexity, especially in the case of clouds where the heterogeneous physical resources are usually virtualized and managed by the hypervisor (Virtual Machine Monitor — VMM) [8].

With the rapid growth of cloud computing, there is a dramatic increase in the number and variety of applications processed on such platforms. They encompass a wide spectrum of sectors and activities, ranging from social media and big data analytics, to healthcare monitoring and financial applications [9,10]. The workload generated by such applications usually comprises jobs

* Corresponding author.
E-mail addresses: gstavrin@csd.auth.gr (G.L. Stavrinides), karatza@csd.auth.gr (H.D. Karatza).

https://doi.org/10.1016/j.future.2019.02.019
0167-739X/© 2019 Elsevier B.V. All rights reserved.

with multiple component tasks that have precedence constraints among them. That is, each job forms a *workflow*, where the output data of a task are used as input by other tasks of the job [11]. A task can start execution only when all of its predecessor tasks have been completed. Timeliness and results quality are amongst the most important QoS requirements commonly defined in a SLA [12]. Typically, each job features a *firm deadline* within which all of its tasks should be completed. Otherwise, a late result would be useless. Such jobs are commonly referred to as *real-time* jobs [13].

In such a real-time setting, it is often more desirable for a job to provide an approximate result by its deadline, than a precise result late. Based on this observation, Lin et al. proposed the *approximate computations* technique [14]. According to this approach, a real-time job is allowed to return intermediate, approximate (imprecise) results of poorer, but still acceptable quality, when its deadline cannot be met. In order to achieve this, each task is decomposed into a *mandatory part*, which provides an approximate result of the minimum acceptable quality, and an *optional part*, which enhances the result provided by the mandatory part. Due to the possibly approximate results of their parent tasks, child tasks in a workflow application may have error in their input data, which may require additional processing time in order to produce an acceptable result [15,16].

1.1. Motivation

The energy, QoS and cost-related challenges involved with processing inherently complex workloads on cloud computing platforms, require the employment of novel and effective scheduling techniques [17]. In order to address these challenges, the orchestration of the workload execution on the virtual machines (VMs) of the cloud should take into account the characteristics of the underlying physical processors, such as per-core DVFS. By leveraging such features, SLA compliance can be provided at a reasonable monetary cost for the end users, while keeping the energy consumption of the resources as low as possible.

1.2. Contribution

Towards this direction, this paper presents an energy-efficient, QoS-aware and cost-effective scheduling strategy for real-time workflow applications in cloud computing systems. The proposed approach utilizes per-core DVFS on the underlying heterogeneous multi-core processors, as well as approximate computations, in order to fill in schedule gaps. At the same time, it takes into account the effects of input error on the processing time of the component tasks. Our goal is to provide timeliness and energy efficiency by trading off result precision, while keeping the result quality of the completed jobs at an acceptable standard and the monetary cost required for the execution of the jobs at a reasonable level.

The proposed scheduling heuristic is compared to two other baseline policies. The work presented in this paper extends our previous work in [18]. Specifically, the proposed scheduling approach is investigated under the impact of various QoS requirements, expressed in the form of result precision thresholds, whereas only one acceptable level of result precision was considered in our previous work. Furthermore, in this study the monetary cost required for the execution of the jobs is also considered.

The remainder of the paper is organized as follows: Section 2 provides a background on processor performance and power states, as well as an overview of related literature. Section 3 presents the system, workload, approximate computations and energy consumption models, as well as the aspects of the employed SLA. Section 4 describes the proposed scheduling heuristic, while Section 5 gives a description of the performance metrics, the experimental setup and analyzes the results of the simulation experiments. Section 6 summarizes and concludes the paper.

2. Background and related work

The *performance states (P-states)* and the *power states (C-states)* of a processor core can be controlled by the operating system or a hypervisor through the Advanced Configuration and Power Interface (ACPI). ACPI is an industry-standard power management interface [19]. P-states, typically expressed as voltage and frequency pairs, can be used to save energy when running workloads on a core. P0 is the P-state that corresponds to the highest performance level, i.e., the highest voltage and the highest frequency (base frequency), and thus the highest power consumption. Higher P-states (P1, P2 etc.) provide lower performance levels, i.e., lower voltage and lower frequency, and thus lower power consumption. A core may transition from one P-state to another via DVFS. Even though the transition between P-states incurs a latency, it is commonly at the scale of microseconds and thus negligible when compared to the execution time of applications, which is usually at the scale of minutes or hours [8,20].

C-states can help save energy when a core is idle. C0 is the active state in which the core is executing instructions at the performance level specified by the utilized P-state. C1 is the first power state in which the core is idle. When a core is in C1, it can switch to C0 virtually without any latency. In this state, the core consumes less power than when active (i.e., when in C0). In higher C-states (C2, C3 etc.) the core is set into deeper sleep states, where more aggressive power conservation techniques are utilized, such as clock gating and power gating. However, the higher the C-state of the core, the more significant becomes the delay to switch back to C0. Therefore, in a dynamic and real-time setting where jobs arrive at the system dynamically and their deadlines must be met, switching to a C-state higher than C1 may cause poor performance [21].

In its general form, the scheduling problem in distributed platforms such as the cloud, concerns the mapping of a set of application tasks to a set of computational resources, in order to complete all tasks under the specified constraints. Completing each task within its deadline, minimizing the response time and tardiness of the tasks, as well as the energy consumption of the computational resources, are some typical scheduling objectives [22]. In its general form, the scheduling problem has been shown to be NP-complete [23].

A large body of research has been focused on energy-efficient scheduling heuristics [24–32]. DVFS is a technique typically used in this context. Wang et al. studied in [33] the slack time for non-critical jobs, extended their execution time and reduced the energy consumption of a DVFS-enabled homogeneous cluster, without increasing the execution time of the tasks as a whole. However, although the jobs featured precedence constraints among their tasks, no deadlines were considered. Mhedheb et al. proposed and implemented in [34] a load-aware and thermal-aware VM scheduling mechanism in a cloud platform, capable of preventing the occurrence of over-loaded or over-heated physical servers. Even though DVFS was utilized for the power management of the physical machines, single-core hosts were considered.

Mizotani et al. proposed in [35] a heuristic that utilized approximate computations with DVFS, for the scheduling of periodic, independent real-time tasks on a homogeneous multiprocessor. According to this approach, the mandatory parts of the tasks had always higher priority for execution than the optional parts. Furthermore, the mandatory part with the earliest deadline had the highest priority for processing. The algorithm utilized the slack time that might occur due to the early completion of a mandatory part, for the scheduling of the optional part of the task at a lower processor speed, using DVFS. However, this approach is not suitable for workflow applications, as it only considers independent tasks. A similar method was proposed by Yu et al. in [36]. The target system under study was a homogeneous multiprocessor, as in the previous method. Even though this approach considered applications with component tasks featuring precedence constraints, no data dependencies (and thus no input error) were considered between the tasks. According to both of these methods, heterogeneous processors with per-core DVFS were not considered.

On the other hand, Lin et al. proposed in [37] scheduling algorithms that leveraged per-core DVFS in a system with heterogeneous processors, in an attempt to achieve a balance between performance and energy consumption. Two scheduling modes were considered: (a) a batch mode, in which tasks were executed in batches, and (b) an online mode, in which tasks with different time constraints, arrival times and computational requirements co-existed in the system. Even though heterogeneous processors and per-core DVFS were considered, the workload consisted of simple independent tasks. Moreover, no approximate computations were utilized. In our previous work in [38], we presented a novel approach for the scheduling of real-time workflow applications in clouds that utilized schedule gaps with approximate computations, where the underlying computing resources where heterogeneous. However, even though our approach took into account the effect of input error on the component tasks of the applications, it did not utilize DVFS or any other energy-aware heuristic.

3. Problem formulation

3.1. System model

The cloud computing system under study has an underlying infrastructure that consists of a set $\mathcal{H} = \{host_1, \ldots, host_h\}$ of h physical hosts with heterogeneous processors. Each host $host_i$ has a multi-core processor that consists of a set $\mathcal{C}_i = \{core_1^i, \ldots, core_{c_i}^i\}$ of c_i cores and supports a set $\mathcal{P}_i = \{(V_{cc_0}^i, f_0^i), \ldots, (V_{cc_{p_i-1}}^i, f_{p_i-1}^i)\}$ of p_i P-states (i.e., voltage and frequency pairs), where V_{cc} is the supply voltage (in volts) and f is the operating frequency (in GHz). The voltage and frequency pair $(V_{cc_0}^i, f_0^i)$ corresponds to the lowest P-state (P0) of the processor, which provides the highest performance level, i.e., the highest operating frequency (base frequency). The pair $(V_{cc_{p_i-1}}^i, f_{p_i-1}^i)$ corresponds to the highest P-state (P$p_i - 1$) of the processor, which provides the lowest performance level, i.e., the lowest operating frequency.

All of the processors support the same instruction set and per-core DVFS. This is a reasonable assumption, as the physical hosts utilized in the data centers of major cloud vendors, such as Amazon Web Services, Microsoft Azure and Google Cloud Platform, typically use processors that support the x86-64 instruction set and per-core DVFS (such as modern AMD and Intel processor architectures). All of the cores of each processor support the same set of P-states and thus the same set of voltage and frequency pairs. Each core can operate at a different P-state from the other cores of the same processor. Different processors may support different sets of P-states. All of the cores require the same number of clock cycles per instruction. There is a set $\mathcal{V} = \{vm_1, \ldots, vm_v\}$ of v VMs in the cloud, where each VM is assigned a virtual CPU (vCPU). Each vCPU corresponds to a physical core. Consequently, the operating frequency f_i of a VM vm_i corresponds to the operating frequency of the assigned physical core, according to its current P-state. The VMs in the cloud are fully connected by a virtual network. The data transfer rate between two VMs vm_i and vm_j is denoted by l_{ij} and is uniformly distributed in the range $\left[\bar{l} \cdot (1 - L/2), \bar{l} \cdot (1 + L/2)\right]$, where L is the heterogeneity degree of the virtual network, whereas $\bar{l}$ is the mean data transfer rate of the communication links. A central scheduler running on a dedicated host is responsible for scheduling the tasks to the VMs in the cloud [39]. The target cloud computing environment is illustrated in Fig. 1.

3.2. Workload model

Real-time jobs corresponding to workflow applications arrive dynamically at the cloud computing environment in a Poisson stream with rate λ. Each job is represented by a *Directed Acyclic Graph (DAG)* $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is the set of the nodes of the graph and $\mathcal{E}$ is the set of the directed edges between the nodes. Each node represents a component task n_i of the job, whereas a directed edge e_{ij} between two tasks n_i and n_j represents the data that must be transferred from task n_i to task n_j. The terms job, workflow and DAG are used interchangeably in the rest of the paper. The component tasks of a workflow are not preemptible, as preemption of real-time tasks may lead to performance degradation [40,41]. Each task n_i has a weight w_i that denotes its *computational volume*, i.e., the number of clock cycles required to execute the instructions of the particular task. The computational volume of each task is exponentially distributed with mean $\overline{w}$. The *computational cost* of the task n_i on a VM vm_j is given by:

$$Comp(n_i, vm_j) = w_i / f_j \tag{1}$$

where f_j is the operating frequency of VM vm_j.

Each edge e_{ij} between two tasks n_i and n_j has a weight z_{ij} that represents its *communication volume*, i.e., the number of GB of data needed to be transferred between the two tasks. The communication volume of each edge is exponentially distributed with mean $\bar{z}$. The *communication cost* of the edge e_{ij} is incurred when data are transferred from task n_i (scheduled on VM vm_m) to task n_j (scheduled on VM vm_n) and is defined as:

$$Comm\left((n_i, vm_m), (n_j, vm_n)\right) = z_{ij} / l_{mn} \tag{2}$$

where l_{mn} is the data transfer rate of the communication link between the VMs vm_m and vm_n. In case both tasks n_i and n_j are scheduled on the same VM or on VMs that run on the same physical host, the communication cost of the edge e_{ij} is considered negligible. The length of a path in the graph is the sum of the computational and communication costs of all of the tasks and edges, respectively, on the path. The *critical path length CPL* is the length of the longest path in the graph. Each real-time workflow has a *firm deadline D* within which all of its component tasks must finish execution. It is given by:

$$D = A + RD \tag{3}$$

where A is the *arrival time* of the workflow and RD is its *relative deadline*, which is uniformly distributed in the range [CPL, $2CPL$].

The *communication to computation ratio CCR* of a workflow is the ratio of its average communication cost to its average computational cost on the target system and is given by:

$$CCR = \frac{\sum_{e_{ij} \in \mathcal{E}} \overline{Comm(e_{ij})}}{\sum_{n_i \in \mathcal{N}} \overline{Comp(n_i)}} \tag{4}$$

where $\mathcal{N}$ and $\mathcal{E}$ are the sets of the nodes and edges of the workflow, respectively. $\overline{Comm(e_{ij})}$ is the average communication cost of the edge e_{ij} over all of the communication links in the system, whereas $\overline{Comp(n_i)}$ is the average computational cost of the task n_i over all of the VMs in the system. It is noted that the base frequency of each vCPU is considered for this calculation. An example of a workflow application is shown in Fig. 2.

3.3. Approximate computations model

The computational volume w_i of each component task n_i of a workflow is assumed to consist of a *mandatory part* mp_i, followed by an *optional part* op_i:

$$w_i = mp_i + op_i \tag{5}$$

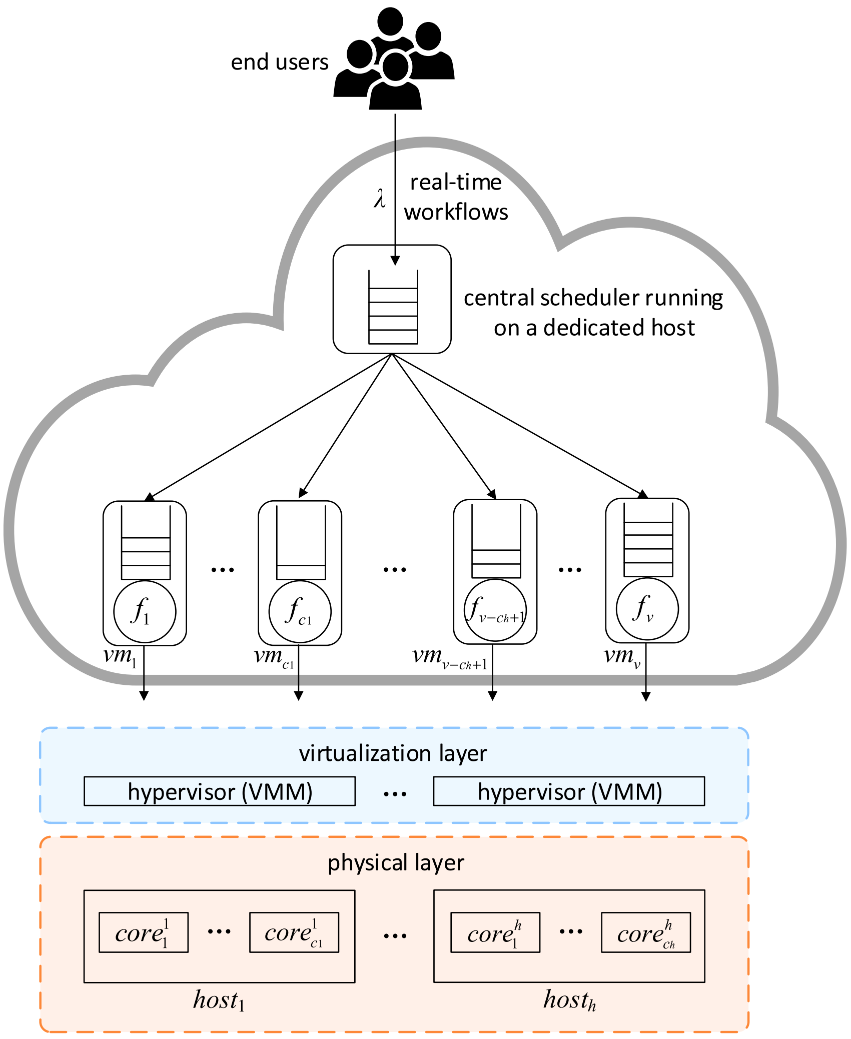

Fig. 1. The cloud computing environment under study.

where $0 \leq mp_i \leq w_i$. A task is completed when at least its mandatory part has been completed. The task can either complete its optional part entirely, partially or it may skip its whole optional part, depending on the decision of the scheduler. The results of a partially completed task n_i are approximate and therefore the task has *output error*, which is given by:

$$OE_i = \frac{\delta_i}{op_i} \tag{6}$$

where δ_i is the discarded fraction of the optional part op_i of the task. Since $0 \leq \delta_i \leq op_i$, from (6) it follows that $0 \leq OE_i \leq 1$.

Since the output data of the task are used as input by its child tasks, the error is propagated to its child tasks as *input error*. It is assumed that the input error of a task n_i is equal to the total output error of its parent tasks:

$$IE_i = \sum_{n_j \in \mathcal{U}_i} OE_j \tag{7}$$

where $\mathcal{U}_i$ is the set of the parent tasks of task n_i. In case a task has input error, there is an impact on its execution time. Specifically, its mandatory part is extended, since more instructions and thus clock cycles are required by the task to handle the error and produce an acceptable result. It is assumed that the optional part of the task is not affected by its input error.

The *mandatory part extension* of a task n_i due to its input error is defined as:

$$mpe_i = mp_i \cdot IE_i \tag{8}$$

where mp_i and IE_i are the mandatory part and the input error of task n_i, respectively. Each task n_i has an *input error limit* IEL_i beyond which the error in its input is not manageable and thus an acceptable result cannot be produced, no matter how much its mandatory part is extended. It is uniformly distributed in the range $[1, u_i]$, where u_i is the number of parent tasks of task n_i. Consequently, the input error IE_i of the task must be $0 \leq IE_i \leq IEL_i$.

3.4. Energy consumption model

The power consumption of a processor core $core_i$ is given by the sum of its dynamic power, which is caused by the switching activity of the transistors, and its static power, which is mainly caused by the leakage current [21,42,43]:

$$P_i = P_{\text{dynamic}_i} + P_{\text{static}_i} \tag{9}$$

The dynamic power of $core_i$ is given by:

$$P_{\text{dynamic}_i} = a_i \cdot C_i \cdot V_{\text{cc}_i}^2 \cdot f_i \tag{10}$$

where a_i is the fraction of transistors that switch at each clock cycle on average, C_i is the transistor capacitance, V_{cc_i} is the supply voltage

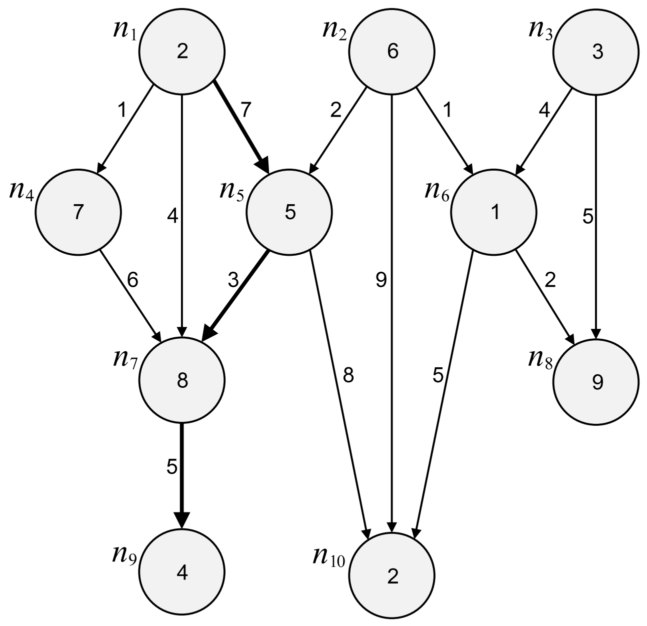

Fig. 2. A workflow application represented as a Directed Acyclic Graph (DAG) with three entry tasks and three exit tasks. The number in each node denotes the average computational cost of the represented task, whereas the number on each edge denotes the average communication cost between the two tasks that it connects. The critical path of the graph is depicted with thick arrows.

and f_i is the operating frequency of the core. The static power of $core_i$ is defined as:

$$P_{\text{static}_i} = I_{\text{leakage}_i} \cdot V_{\text{cc}_i} \tag{11}$$

where I_{leakage_i} is the leakage current.

Hence, the dynamic power of $core_i$ can be expressed as:

$$P_{\text{dynamic}_i} = K_{\text{dynamic}_i} \cdot V^2_{\text{cc}_i} \cdot f_i \tag{12}$$

where K_{dynamic_i} is the *dynamic power factor* of the core. The static power of the core is typically proportional to its dynamic power [44]. Therefore, it can be expressed as:

$$P_{\text{static}_i} = K_{\text{static}_i} \cdot P_{\text{dynamic}_i} \tag{13}$$

where K_{static_i} is the *static power factor* of the core.

Consequently, from (9), (12) and (13) it follows that:

$$P_i = (1 + K_{\text{static}_i}) \cdot K_{\text{dynamic}_i} \cdot V^2_{\text{cc}_i} \cdot f_i \tag{14}$$

It is assumed that only the C-states C0 and C1 are utilized, due to the real-time requirements of the workload. C1 corresponds to the static power consumption of the core, at its lowest possible voltage level. Hence, the energy consumption of a core $core_i$ over a period of time $[t_0, t_1]$ can be defined as an integral of its power consumption [45]:

$$E_i = \int_{t_0}^{t_1} P_i(t) \cdot dt \tag{15}$$

3.5. SLA aspects

3.5.1. QoS requirements

The following QoS requirements are imposed by the employed SLA:

- The deadline of each job must be met. In case a job misses its deadline, a SLA violation occurs.
- Each completed job must provide results of acceptable quality.

It is assumed that there is a *result precision threshold RPT*, under which the results of a job are not acceptable. The relation between the result precision threshold and the mandatory part of a task n_i is given by:

$$mp_i = RPT \cdot w_i \tag{16}$$

where w_i is the computational volume of the task. The *result precision* of the task is defined as:

$$RP_i = RPT + (1 - RPT)(1 - OE_i) \tag{17}$$

where OE_i is the output error of the task. The result precision of a job is considered to be equal to the average result precision of its exit tasks.

In case the deadline of an uncompleted job is reached, the job is not always considered lost. Specifically, if the uncompleted tasks of the job are all exit tasks and all of them have completed their mandatory part, then the job is considered completed. In this case, even though the results of the job are approximate, they are still of acceptable quality.

3.5.2. Pricing scheme

We consider a pay-as-you-go pricing scheme, where an hourly rate is charged for each VM utilized for the execution of the workload.

4. Scheduling strategy

In order to schedule the ready tasks waiting in the queue of the central scheduler, a heuristic is employed, which consists of two phases: (a) a *task selection phase* and (b) a *VM selection phase*.

4.1. Task selection phase

Tasks are prioritized according to their job's deadline. The task that belongs to the job with the earliest deadline, is selected first by the scheduler. Consequently, tasks are prioritized according to the *Earliest Deadline First (EDF)* policy. In case two or more ready tasks have the same priority, the task with the largest average computational cost is selected first. This tie-breaking technique is employed, as it has been proven in the literature that when combined with EDF, it gives better results than other methods, such as random selection [46].

4.2. VM selection phase

Once a task is selected by the scheduler, it is allocated to the VM that can provide it with the earliest *estimated finish time EFT*. The *EFT* of a ready task n_i on a VM vm_k is given by:

$$EFT(n_i, vm_k) = \max\{t_{\text{data}}(n_i, vm_k), t_{\text{idle}}(n_i, vm_k)\} + Comp(n_i, vm_k) \tag{18}$$

where $t_{\text{data}}(n_i, vm_k)$ is the time at which all input data of task n_i will be available on VM vm_k, whereas $t_{\text{idle}}(n_i, vm_k)$ is the time at which vm_k will be able to execute task n_i.

In order to calculate the term $t_{\text{idle}}(n_i, vm_k)$, we first determine the position that task n_i would be placed in the queue of VM vm_k. This position is determined by taking into account the task's priority and by utilizing possible schedule gaps, using DVFS and approximate computations. The following steps are performed:

- **Step 1:** We first find the position at which the ready task n_i would be placed in the VM's queue, according to its priority.

- **Step 2:** In case all of the required input data of the ready task n_i are available on VM vm_k (i.e., $t_{\text{data}}(n_i, vm_k) = t_{\text{current}}$, where t_{current} is the current time), we check whether a schedule gap exists. A schedule gap is formed when the VM is idle and the task n_q placed at the head of the queue is still in the process of receiving its required input data from other hosts. The capacity g of the schedule gap is calculated as: $g = t_{\text{data}}(n_q, vm_k) - t_{\text{current}}$, where $t_{\text{data}}(n_q, vm_k)$ is the time at which all of the required input data of task n_q will be received.
- **Step 3:** If a schedule gap exists, we calculate the maximum possible discarded fraction $\delta_{\max_i}$ of the ready task's optional part, so that the input error limit of each of its child tasks is not exceeded. This is given by:

$$\delta_{\max_i} = op_i \cdot \min\left\{1, \min_{n_j \in \mathcal{Y}_i}\left\{IEL_j - IE_j\right\}\right\} \tag{19}$$

  where $\mathcal{Y}_i$ is the set of child tasks of the ready task n_i. In case the examined ready task is an exit task of its job, then we consider that $\delta_{\max_i} = 0$, since all exit tasks should be allowed to produce precise results.
- **Step 4:** Subsequently, we try to fill in the schedule gap with the maximum possible part of ready task n_i, using the minimum possible supported operating frequency of VM vm_k. In case the whole task cannot be inserted into the schedule gap, we insert only a part of it, utilizing approximate computations. Specifically, for each supported operating frequency f_k of VM vm_k, we check the following conditions, starting from the lowest supported frequency (i.e., the highest P-state of the corresponding core):

$$g \geq w_i / f_k \tag{20}$$

$$g \geq (w_i - \delta_{\max_i}) / f_k \tag{21}$$

$$\delta_{\max_i} \cdot P_k \geq \sum_{n_j \in \mathcal{Y}_i} \frac{\delta_{\max_i}}{op_i} \cdot mp_j \cdot \overline{P} \tag{22}$$

  In condition (22) we check whether the energy that can be saved by skipping a fraction of the ready task n_i's optional part equal to $\delta_{\max_i}$, is greater than or equal to the total average additional energy that would be consumed by the extended mandatory part of n_i's child tasks, due to n_i's potential output error. P_k is the power consumption of the corresponding physical core of VM vm_k for the particular frequency level, whereas $\overline{P}$ is the average power consumption of all of the cores in the system, operating at base frequency. In case condition (20) holds, the whole task can be inserted into the schedule gap and the VM can process it with frequency f_k. In case conditions (21) and (22) hold, only a part of the task can be inserted into the schedule gap and the VM can process it with frequency f_k. The part of the task that would be processed in this case would be equal to its computational volume w_i' that fits into the gap at the particular frequency, i.e., $w_i' = g \cdot f_k$. In case the ready task n_i cannot be placed into a schedule gap or a schedule gap does not exist, the position of task n_i in vm_k's queue is determined only by its priority (i.e., as in step 1).

The pseudocode for determining whether a schedule gap can be utilized as described above (steps 2–4), is given in Algorithm 1. The utilization of schedule gaps is also performed in the same manner for a task waiting in a queue when all of its input data become available on its assigned VM. Furthermore, it is also performed for tasks that are waiting for service in a queue and either the task in service completes execution or a task is discarded from the queue because its job's deadline has been reached. In the last two cases, eligible tasks are considered according to their priority.

Algorithm 1 Schedule gap utilization with DVFS and approximate computations.

Input: A ready task n_i and a VM vm_k.
Output: Decision on whether a gap in vm_k's schedule can be utilized by task n_i.

```
1: gapExists ← false
2: gapIsUtilized ← false
3: if $t_{\text{data}}(n_i, vm_k) = t_{\text{current}}$ then
4:     if $vm_k$ is idle and $t_{\text{data}}(n_q, vm_k) > t_{\text{current}}$ then
5:         gapExists ← true
6:         $g \leftarrow t_{\text{data}}(n_q, vm_k) - t_{\text{current}}$
7:     end if
8: end if
9: if gapExists then
10:    if $n_i$ is an exit task then
11:        $\delta_{\max_i} \leftarrow 0$
12:    else
13:        $\delta_{\max_i} \leftarrow op_i \cdot \min\{1, \min_{n_j \in \mathcal{Y}_i}\{IEL_j - IE_j\}\}$
14:    end if
15:    for each $f_{k_p} \in \mathcal{P}_k$ starting from $f_{k_{\min}}$ do
16:        if $g \geq w_i / f_{k_p}$ then
17:            gapIsUtilized ← true
18:            $w_i' \leftarrow w_i$
19:            selectedFrequency ← $f_{k_p}$
20:            return gapIsUtilized
21:        else if $g \geq (w_i - \delta_{\max_i})/f_{k_p}$ and
           $\delta_{\max_i} \cdot P_{k_p} \geq \sum_{n_j \in \mathcal{Y}_i} \frac{\delta_{\max_i}}{op_i} \cdot mp_j \cdot \overline{P}$ then
22:            gapIsUtilized ← true
23:            $w_i' \leftarrow g \cdot f_{k_p}$
24:            selectedFrequency ← $f_{k_p}$
25:            return gapIsUtilized
26:        end if
27:    end for
28: end if
29: return gapIsUtilized
```

The proposed scheduling approach is energy-efficient through the utilization of per-core DVFS during the VM selection phase. Furthermore, it is QoS-aware, as the deadlines of the jobs are taken into account during the task selection phase, using the EDF algorithm. Moreover, it is cost-effective, as it tries to minimize the idle time of the VMs (and thus the associated monetary cost required for running each VM), through the utilization of schedule gaps during the VM selection phase. The provided energy efficiency, QoS and cost effectiveness are further enhanced through the combination of DVFS with approximate computations, which allows the partial execution of the tasks inserted into schedule gaps, utilizing lower frequencies. Consequently, less energy is required for the execution of the tasks, more jobs are completed within their deadline and less time and thus monetary cost is required for executing the workload on the VMs. These objectives are achieved, while ensuring at the same time that all of the jobs will provide results of acceptable quality.

We refer to our proposed scheduling heuristic as *Earliest Deadline First with DVFS and Approximate Computations (EDF_DVFS_AC)*. The proposed approach is compared to two baseline scheduling policies: (a) EDF, which is not energy-aware and it does not utilize any schedule gaps, DVFS or approximate computations and (b) EDF_DVFS, which is a modified version of our proposed heuristic, where schedule gaps are utilized using per-core DVFS (i.e., it is energy-aware), but only if the whole task fits into the gap (i.e., approximate computations are not used).

5. Performance evaluation

5.1. Performance metrics

The following metrics were employed for the evaluation of the performance of our proposed scheduling algorithm:

- *Total energy consumption*, which is the total energy in kilowatt-hours (kWh) consumed by all of the cores of the system under study, during the observed time period.
- *SLA violation ratio*, which is the ratio of the number of jobs that did not finish their execution within their deadline (and thus lost), over the number of all of the jobs that arrived at the central scheduler of the cloud, during the observed time period.
- *Average result precision*, which is the average result precision of the completed jobs, during the observed time period.
- *Total monetary cost*, which is the total monetary cost in US dollars ($) required for the completion of the same number of jobs, under each scheduling policy.

5.2. Experimental setup

The performance of the system was evaluated by conducting a series of simulation runs, rather than by analytical methods. Due to the complexity of the system and workload models under study, analytical modeling would be very difficult to implement and would require several simplifying assumptions that would have an unpredictable impact on the results. In order to model in detail all of the characteristics of the framework under study and have full control on all of its parameters, we chose not to use one of the available simulation packages. Instead, we implemented our own discrete-event simulation program in C++, tailored to the specific requirements of the particular problem, using the independent replications method. Furthermore, synthetic workload was used, in order to obtain unbiased, general results, not applicable only to particular workload traces. The workflows were generated randomly, using our own random DAG generator, as described in [15]. Each generated DAG was a weakly connected graph, having a path between any pair of tasks, without taking into account the direction of the edges. There was at least one entry and one exit task in each generated DAG.

For comparison purposes, the P-states of the host processors were based on those of real-world processors. Specifically, half of the hosts in the underlying infrastructure were considered to be small in terms of the number of processor cores, having per-core P-states modeled after the Intel Xeon E5450 processor, whereas the other half of the hosts were considered to be large, having per-core P-states modeled after the Intel Xeon X5670 processor. The supported P-states of the small and large host processor cores are shown in Table 1 [47,48]. The static power factor of the small and large host processor cores was chosen to be $K_{\text{static}} = 0.3$, since the static power consumption in CMOS circuits is approximately 30% of the dynamic power consumption [44]. The dynamic power factor of the cores was calculated based on the maximum power consumption of the employed real-world processors, as provided by the manufacturer [49].

We conducted a set of experiments for each combination of values of the communication to computation ratio (*CCR*) and the result precision threshold (*RPT*) parameters. In terms of the *CCR* parameter, we considered: (a) computationally intensive ($CCR = 0.5$), (b) moderate ($CCR = 1$) and (c) communication intensive ($CCR = 2$) jobs. In terms of the *RPT* parameter, we considered a wide range of values. On the one end of the spectrum, the result precision threshold and thus the mandatory part of each task was considered equal to zero ($RPT = 0$). Consequently, each task could essentially be omitted. However, this was restricted by the imposed input error limit, as well as the requirement that each exit task should always give results of the highest possible precision. On the other end of the spectrum, the result precision threshold was considered equal to 1 ($RPT = 1$), meaning that each task was not allowed to omit any fraction of its computational volume and it was thus forced to always produce precise results (i.e., approximate computations were not utilized in this case). Between these two extreme cases, the following values were also considered for the result precision threshold: $RPT = \{0.25, 0.5, 0.75\}$. In these cases, a task was allowed to omit a large, medium or small fraction of its computational volume, respectively. This variety of result precision thresholds enabled us to investigate our proposed scheduling heuristic under the impact of various QoS requirements, in contrast to our previous work in [18], where a fixed value for the result precision threshold was considered.

Table 1
P-states of host processor cores.

P-state	Voltage (V)	Frequency (GHz)
Small host processor core		
P0	1.35	3.00
P1	1.17	2.67
P2	1.00	2.33
P3	0.85	2.00
Large host processor core		
P0	1.35	2.93
P1	1.29	2.80
P2	1.23	2.67
P3	1.17	2.54
P4	1.11	2.40
P5	1.05	2.27
P6	0.99	2.14
P7	0.92	2.00
P8	0.86	1.87
P9	0.80	1.74
P10	0.75	1.60

For computationally intensive DAGs ($CCR = 0.5$), the mean computational volume of the tasks was selected to be equal to $\overline{w} = 1.758 \cdot 10^{12}$ clock cycles, so that on average, a task would take 10 min to execute on a vCPU corresponding to a core of a large processor, operating at base frequency (i.e., 2.93 GHz). For moderate and communication intensive DAGs ($CCR = \{1, 2\}$), the mean computational volume of the tasks was selected to be equal to $\overline{w} = 0.879 \cdot 10^{12}$ clock cycles, so that on average, a task would take half the time to execute compared to the computationally intensive case (i.e., 5 min). For each *CCR* and mean computational volume $\overline{w}$, the mean communication volume $\overline{z}$ was calculated from (4). In order for the system to be stable, the job arrival rate was chosen to be $\lambda = 0.004$. The hourly rate per VM was considered to be $0.019, which is the average hourly rate charged by Amazon EC2 for on-demand VM instances with one vCPU and variable EC2 Compute Units (for Windows usage) [50]. The input parameters of the simulation model are shown in Table 2.

We ran 30 replications of the simulation with different seeds of random numbers, for each set of input parameters. Each replication was terminated when 4000 jobs had been completed. We found by experimentation that this simulation run length was long enough to minimize the effects of warm-up time. For every mean value, a 95% confidence interval was calculated. The half-widths of all of the confidence intervals were less than 5% of their respective mean values. Furthermore, in order to examine whether the differences between the mean values obtained by each scheduling method were statistically significant, a 95% confidence interval was calculated for the difference between each pair of mean values. The calculated confidence intervals did not include 0 and thus the differences in the results between the employed scheduling policies were statistically significant.

5.3. Simulation results and discussion

The simulation results for the total energy consumption, SLA violation ratio and total monetary cost metrics, are presented in Figs. 3–5, respectively, for each (*CCR*, *RPT*) pair of values, under

Table 2
Input parameters of the simulation model.

Parameter	Value
System model parameters	
Small hosts:	
Number of hosts (i.e., processors)	$h_{\text{small}} = 20$
Number of host processor cores	$c_{\text{small}} = 4$
Number of core P-states	$p_{\text{small}} = 4$
Static power factor of processor core	$K_{\text{static}_{\text{small}}} = 0.3$
Dynamic power factor of processor core	$K_{\text{dynamic}_{\text{small}}} = 2.8154 \cdot 10^{-9}$
Large hosts:	
Number of hosts (i.e., processors)	$h_{\text{large}} = 20$
Number of host processor cores	$c_{\text{large}} = 6$
Number of core P-states	$p_{\text{large}} = 11$
Static power factor of processor core	$K_{\text{static}_{\text{large}}} = 0.3$
Dynamic power factor of processor core	$K_{\text{dynamic}_{\text{large}}} = 2.2846 \cdot 10^{-9}$
Cloud:	
Number of VMs	$v = 200$
Heterogeneity degree of virtual network	$L = 0.5$
Mean link data transfer rate	$\bar{l} = 1$ Gbps
Hourly rate per VM	\$0.019
Workload model parameters	
Number of completed DAGs	4000
DAG arrival rate	$\lambda = 0.004$
Number of tasks per DAG	$n \sim U[16, 128]$
DAG communication to computation ratio	$CCR = \{0.5, 1, 2\}$
Mean task computational volume	$\overline{w} = 0.879 \cdot 10^{12}$ clock cycles for $CCR = \{1, 2\}$
	$\overline{w} = 1.758 \cdot 10^{12}$ clock cycles for $CCR = 0.5$
Result precision threshold	$RPT = \{0, 0.25, 0.5, 0.75, 1\}$
Other model parameters	
Scheduling policies	EDF, EDF_DVFS, EDF_DVFS_AC

each scheduling policy, EDF, EDF_DVFS and EDF_DVFS_AC. The comparison of the scheduling strategies with respect to all of the employed performance metrics is shown in Tables 3–6. It is demonstrated that for all types of workload and result precision thresholds, the proposed scheduling heuristic, EDF_DVFS_AC, outperformed the other two baseline policies, providing the lowest total energy consumption, SLA violations ratio and total monetary cost. This was achieved with only an insignificant loss of job result precision. Specifically, in the worst case, the proposed approach provided average result precision equal to 0.9918, which was only about 0.82% lower than the precise results provided by the other two policies.

It can be observed that for all types of workload – computationally intensive ($CCR = 0.5$), moderate ($CCR = 1$) and communication intensive ($CCR = 2$) workflows – EDF_DVFS_AC provided lower energy consumption, SLA violations and monetary cost when lower result precision thresholds were employed. This was due to the fact that lower precision thresholds allowed the component tasks of the workflows to omit larger fractions of their computational volume. Consequently, more schedule gaps were utilized, since it was more likely for a partial task to fit into a gap.

The two baseline policies, EDF and EDF_DVFS, were not affected by the variation in the values of RPT, since they did not utilize approximate computations. For the highest precision threshold ($RPT = 1$), EDF_DVFS_AC exhibited the same performance as EDF_DVFS, since in that case both strategies filled in schedule gaps using only DVFS, without utilizing approximate computations. On the other hand, lower result precision thresholds favored the proposed heuristic. It can also be observed that EDF_DVFS_AC did not provide lower average result precision when lower precision thresholds were employed. This was due to the imposed input error limit on the tasks, as well as the requirement that each exit task should always give results of the highest possible precision.

The average energy savings in the case of EDF_DVFS_AC were 11.01% compared to the non energy-aware policy, EDF, and 9.20% compared to the other energy-aware scheduling method, EDF_DVFS. The average improvement in SLA violations was 33.98% and 27.90%, compared to EDF and EDF_DVFS, respectively. In terms of the result precision, the average decrease was 0.21% compared to both baseline strategies. Furthermore, EDF_DVFS_AC provided 7.82% average monetary cost savings compared to EDF, and 5.70% compared to EDF_DVFS. On average, the energy and monetary cost savings were more significant in the case of computationally intensive workflows, which required in general longer execution times and thus greater energy consumption and higher monetary cost. Therefore, with the effective utilization of schedule gaps, the execution time of the workflows shrunk, since partial tasks could be placed in the gaps using approximate computations. At the same time, less energy was required to execute them, not only due to their smaller execution times, but also due to the utilization of lower operating frequencies through the application of DVFS on the physical cores.

On the other hand, the improvement in the SLA violations ratio was on average more significant in the case of communication intensive workflows. Due to the longer data transfer times, more gaps formed in the schedule of the VMs. Therefore, the effective utilization of those gaps with our proposed approach, which provided flexibility by allowing the insertion of partial tasks into the gaps, yielded better performance. The result precision decrease was on average greater in the case of computationally intensive workflows, which indicates that more exit tasks were forced to discard a fraction of their optional part in order to meet their job's deadline, compared to the case of moderate and communication intensive workflows.

6. Conclusions and future work

In this paper, we proposed an energy-efficient, QoS-aware and cost-effective scheduling strategy for real-time workflow applications in cloud computing systems. Our approach utilized per-core DVFS on the underlying heterogeneous multi-core processors, as well as approximate computations, in order to fill in schedule gaps,

Table 3
Energy savings (%).

(*CCR*, *RPT*)	EDF_DVFS vs. EDF	EDF_DVFS_AC vs. EDF	EDF_DVFS_AC vs. EDF_DVFS
(0.5,0)	3.93	43.70	41.40
(0.5,0.25)	3.93	16.76	13.36
(0.5,0.5)	3.93	12.11	8.52
(0.5,0.75)	3.93	8.74	5.01
(0.5,1)	3.93	3.93	0.00
(1,0)	0.20	21.01	20.85
(1,0.25)	0.20	5.03	4.83
(1,0.5)	0.20	3.24	3.05
(1,0.75)	0.20	1.54	1.34
(1,1)	0.20	0.20	0.00
(2,0)	2.02	27.01	25.50
(2,0.25)	2.02	10.18	8.33
(2,0.5)	2.02	5.60	3.65
(2,0.75)	2.02	4.10	2.12
(2,1)	2.02	2.02	0.00
Average	2.05	11.01	9.20

Table 4
SLA violations decrease (%).

(*CCR*, *RPT*)	EDF_DVFS vs. EDF	EDF_DVFS_AC vs. EDF	EDF_DVFS_AC vs. EDF_DVFS
(0.5,0)	6.90	90.91	90.23
(0.5,0.25)	6.90	34.53	29.68
(0.5,0.5)	6.90	23.50	17.83
(0.5,0.75)	6.90	16.76	10.60
(0.5,1)	6.90	6.90	0.00
(1,0)	2.08	77.40	76.92
(1,0.25)	2.08	31.89	30.44
(1,0.5)	2.08	28.12	26.60
(1,0.75)	2.08	14.78	12.97
(1,1)	2.08	2.08	0.00
(2,0)	15.84	76.33	71.88
(2,0.25)	15.84	31.95	19.14
(2,0.5)	15.84	30.91	17.91
(2,0.75)	15.84	27.82	14.24
(2,1)	15.84	15.84	0.00
Average	8.27	33.98	27.90

Table 5
Result precision decrease (%).

(*CCR*, *RPT*)	EDF_DVFS vs. EDF	EDF_DVFS_AC vs. EDF	EDF_DVFS_AC vs. EDF_DVFS
(0.5,0)	0.00	0.34	0.34
(0.5,0.25)	0.00	0.82	0.82
(0.5,0.5)	0.00	0.42	0.42
(0.5,0.75)	0.00	0.14	0.14
(0.5,1)	0.00	0.00	0.00
(1,0)	0.00	0.25	0.25
(1,0.25)	0.00	0.33	0.33
(1,0.5)	0.00	0.17	0.17
(1,0.75)	0.00	0.05	0.05
(1,1)	0.00	0.00	0.00
(2,0)	0.00	0.22	0.22
(2,0.25)	0.00	0.27	0.27
(2,0.5)	0.00	0.09	0.09
(2,0.75)	0.00	0.03	0.03
(2,1)	0.00	0.00	0.00
Average	0.00	0.21	0.21

Table 6
Monetary cost savings (%).

(*CCR*, *RPT*)	EDF_DVFS vs. EDF	EDF_DVFS_AC vs. EDF	EDF_DVFS_AC vs. EDF_DVFS
(0.5,0)	4.32	34.57	31.61
(0.5,0.25)	4.32	16.67	12.90
(0.5,0.5)	4.32	12.35	8.39
(0.5,0.75)	4.32	9.20	5.10
(0.5,1)	4.32	4.32	0.00
(1,0)	0.89	7.14	6.31
(1,0.25)	0.89	3.57	2.70
(1,0.5)	0.89	3.46	2.59
(1,0.75)	0.89	2.08	1.20
(1,1)	0.89	0.89	0.00
(2,0)	1.74	9.57	7.96
(2,0.25)	1.74	4.35	2.65
(2,0.5)	1.74	3.99	2.29
(2,0.75)	1.74	3.48	1.77
(2,1)	1.74	1.74	0.00
Average	2.32	7.82	5.70

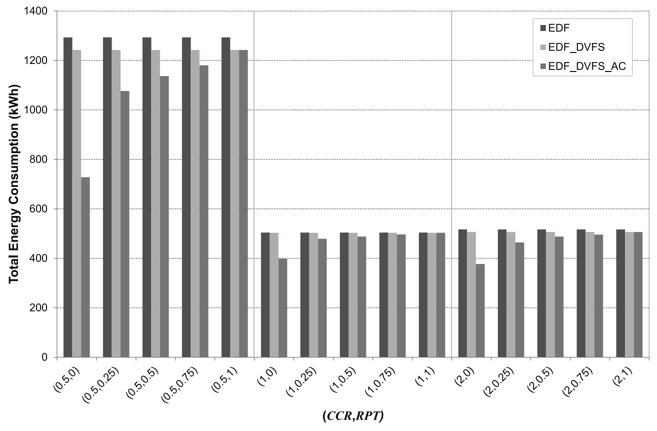

Fig. 3. Total energy consumption (kWh) for each pair of (*CCR*, *RPT*) values.

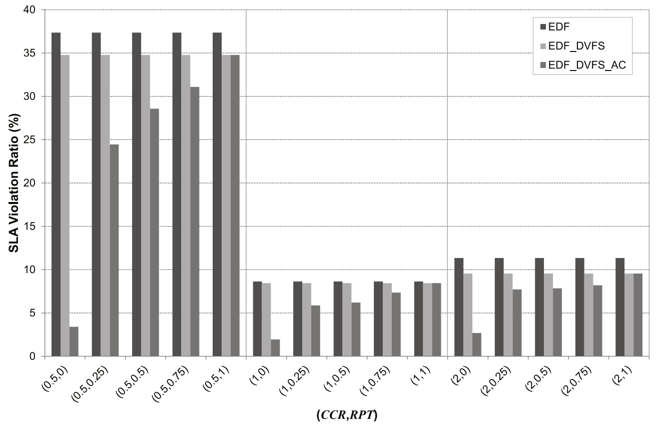

Fig. 4. SLA violation ratio (%) for each pair of (*CCR*, *RPT*) values.

taking into account the effects of input error on the processing time of the component tasks. We conducted a series of simulation experiments under various QoS requirements, expressed in the form of result precision thresholds, where the proposed heuristic, EDF_DVFS_AC, outperformed the other two baseline scheduling policies, EDF and EDF_DVFS. In all sets of experiments, it consistently provided not only the lowest energy consumption, but the lowest SLA violations ratio and total monetary cost as well, with only an insignificant loss of job result precision. Our future work plans include the investigation of the proposed scheduling technique in cases where the employed SLA is more relaxed, so that various degrees of tardiness are tolerated in the execution of the jobs.

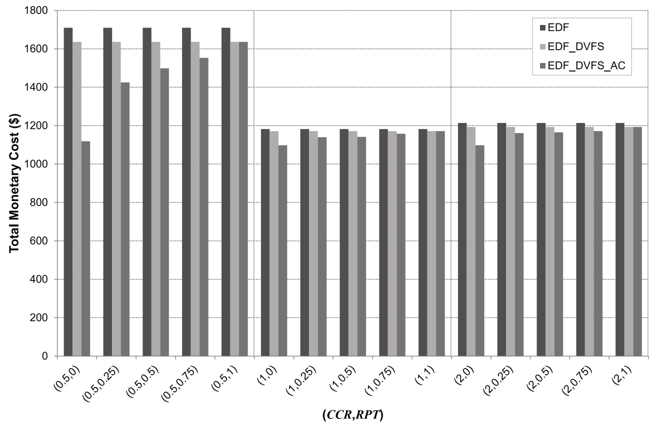

Fig. 5. Total monetary cost ($) for each pair of (*CCR*, *RPT*) values.

References

[1] G.L. Stavrinides, H.D. Karatza, Performance evaluation of a SaaS cloud under different levels of workload computational demand variability and tardiness bounds, Simul. Model. Pract. Theory 91 (2019) 1–12, http://dx.doi.org/10.1016/j.simpat.2018.11.006.

[2] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768, http://dx.doi.org/10.1016/j.future.2011.04.017.

[3] G.L. Valentini, W. Lassonde, S.U. Khan, N.M. Allah, S.A. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej, H. Li, A.Y. Zomaya, C.Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J.E. Pecero, D. Kliazovich, P. Bouvry, An overview of energy efficiency techniques in cluster computing systems, Cluster Comput. 16 (1) (2013) 3–15, http://dx.doi.org/10.1007/s10586-011-0171-x.

[4] G.L. Stavrinides, H.D. Karatza, The impact of workload variability on the energy efficiency of large-scale heterogeneous distributed systems, Simul. Model. Pract. Theory 89 (2018) 135–143, http://dx.doi.org/10.1016/j.simpat.2018.09.013.

[5] J.J. Chen, C.Y. Yang, T.W. Kuo, Slack reclamation for real-time task scheduling over dynamic voltage scaling multiprocessors, in: Proceedings of the 2006 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC'06), 2006, pp. 358–365, http://dx.doi.org/10.1109/SUTC.2006.1636201.

[6] S. Kim, H. Eom, H.Y. Yeom, S.L. Min, Energy-centric DVFS controlling method for multi-core platforms, Computing 96 (12) (2014) 1163–1177, http://dx.doi.org/10.1007/s00607-013-0369-2.

[7] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, R. Geyer, An energy efficiency feature survey of the Intel Haswell processor, in: Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium (IPDPS'15), 2015, pp. 896–904, http://dx.doi.org/10.1109/IPDPSW.2015.70.

[8] R.N. Calheiros, R. Buyya, Energy-efficient scheduling of urgent bag-of-tasks applications in clouds through DVFS, in: Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom'14), 2014, pp. 342–349, http://dx.doi.org/10.1109/CloudCom.2014.20.

[9] I.A.T. Hashem, I. Yaqoob, N.B. Anuar, S. Mokhtar, A. Gani, S.U. Khan, The rise of big data on cloud computing: review and open research issues, Inf. Syst. 47 (2015) 98–115, http://dx.doi.org/10.1016/j.is.2014.07.006.

[10] G.L. Stavrinides, H.D. Karatza, Scheduling data-intensive workloads in large-scale distributed systems: trends and challenges, in: Modeling and Simulation in HPC and Cloud Systems, first ed., in: Studies in Big Data, vol. 36, Springer, 2018, pp. 19–43, http://dx.doi.org/10.1007/978-3-319-73767-6_2, (Chapter 2).

[11] Y. Chen, W.T. Tsai, Service-Oriented Computing and Web Software Integration: From Principles to Development, fifth ed., Kendall Hunt Publishing, 2015.

[12] G.L. Stavrinides, H.D. Karatza, Scheduling real-time jobs in distributed systems - simulation and performance analysis, in: Proceedings of the 1st International Workshop on Sustainable Ultrascale Computing Systems (NESUS'14), 2014, pp. 13–18.

[13] G.L. Stavrinides, H.D. Karatza, Scheduling real-time bag-of-tasks applications with approximate computations in SaaS clouds, Concurr. Comput.: Pract. Exper. (2017) e4208, http://dx.doi.org/10.1002/cpe.4208.

[14] K.J. Lin, S. Natarajan, J.W.S. Liu, Imprecise results: utilizing partial computations in real-time systems, in: Proceedings of the 8th IEEE Real-Time Systems Symposium (RTSS'87), 1987, pp. 210–217.

[15] G.L. Stavrinides, H.D. Karatza, The impact of input error on the scheduling of task graphs with imprecise computations in heterogeneous distributed real-time systems, in: Proceedings of the 18th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA'11), 2011, pp. 273–287, http://dx.doi.org/10.1007/978-3-642-21713-5_20.

[16] G.L. Stavrinides, H.D. Karatza, Scheduling real-time DAGs in heterogeneous clusters by combining imprecise computations and bin packing techniques for the exploitation of schedule holes, Future Gener. Comput. Syst. 28 (7) (2012) 977–988, http://dx.doi.org/10.1016/j.future.2012.03.002.

[17] Y. Chen, Service-Oriented Computing and System Integration: Software, IoT, Big Data, and AI as Services, sixth ed., Kendall Hunt Publishing, 2018.

[18] G.L. Stavrinides, H.D. Karatza, Energy-aware scheduling of real-time workflow applications in clouds utilizing DVFS and approximate computations, in: Proceedings of the IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud'18), 2018, pp. 33–40, http://dx.doi.org/10.1109/FiCloud.2018.00013.

[19] VMware, Host power management in VMware vSphere 5.5, Tech. Rep. EN-001262-00, VMware Inc. (Aug. 2013).

[20] A. Mazouz, A. Laurent, B. Pradelle, W. Jalby, Evaluation of CPU frequency transition latency, Comput. Sci. - Res. Dev. 29 (3) (2014) 187–195, http://dx.doi.org/10.1007/s00450-013-0240-x.

[21] R. Schöne, D. Molka, M. Werner, Wake-up latencies for processor idle states on current x86 processors, Comput. Sci. - Res. Dev. 30 (2) (2015) 219–227, http://dx.doi.org/10.1007/s00450-014-0270-z.

[22] J. Kolodziej, Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems, Springer, 2012.

[23] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, 1979.

[24] Z. Li, J. Ge, H. Hu, W. Song, H. Hu, B. Luo, Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds, IEEE Trans. Serv. Comput. 11 (4) (2015) 713–726, http://dx.doi.org/10.1109/TSC.2015.2466545.

[25] Y. Govindaraju, H. Duran-Limon, A QoS and energy aware load balancing and resource allocation framework for IaaS cloud providers, in: Proceedings of the 9th IEEE/ACM International Conference on Utility and Cloud Computing (UCC'16), 2016, pp. 410–415, http://dx.doi.org/10.1145/2996890.3007895.

[26] T. Kaur, I. Chana, Energy aware scheduling of deadline-constrained tasks in cloud computing, Cluster Comput. 19 (2) (2016) 679–698, http://dx.doi.org/10.1007/s10586-016-0566-9.

[27] H. Duan, C. Chen, G. Min, Y. Wu, Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems, Future Gener. Comput. Syst. 74 (2017) 142–150, http://dx.doi.org/10.1016/j.future.2016.02.016.

[28] F. Juarez, J. Ejarque, R.M. Badia, Dynamic energy-aware scheduling for parallel task-based application in cloud computing, Future Gener. Comput. Syst. 78 (1) (2018) 257–271, http://dx.doi.org/10.1016/j.future.2016.06.029.

[29] S. Kumar, M. Kalra, A hybrid approach for energy-efficient task scheduling in cloud, in: Proceedings of the 2nd International Conference on Communication, Computing and Networking (ICCCN'18), 2018, pp. 1011–1019, http://dx.doi.org/10.1007/978-981-13-1217-5_99.

[30] W. Zhang, Y. Wen, Energy-efficient task execution for application as a general topology in mobile cloud computing, IEEE Trans. Cloud Comput. 6 (3) (2018) 708–719, http://dx.doi.org/10.1109/TCC.2015.2511727.

[31] L. Gu, J. Cai, D. Zeng, Y. Zhang, H. Jin, W. Dai, Energy efficient task allocation and energy scheduling in green energy powered edge computing, Future Gener. Comput. Syst. 95 (2019) 89–99, http://dx.doi.org/10.1016/j.future.2018.12.062.

[32] X. Zhou, G. Zhang, J. Sun, J. Zhou, T. Wei, S. Hu, Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT, Future Gener. Comput. Syst. 93 (2019) 278–289, http://dx.doi.org/10.1016/j.future.2018.10.046.

[33] L. Wang, S.U. Khan, D. Chen, J. Kolodziej, R. Ranjan, C. Xu, A. Zomaya, Energy-aware parallel task scheduling in a cluster, Future Gener. Comput. Syst. 29 (7) (2013) 1661–1670, http://dx.doi.org/10.1016/j.future.2013.02.010.

[34] Y. Mhedheb, F. Jrad, J. Tao, J. Zhao, J. Kolodziej, A. Streit, Load and thermal-aware VM scheduling on the cloud, in: Proceedings of the 13th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'13), 2013, pp. 101–114, http://dx.doi.org/10.1007/978-3-319-03859-9_8.

[35] K. Mizotani, Y. Hatori, Y. Kumura, M. Takasu, H. Chishiro, N. Yamasaki, An integration of imprecise computation model and real-time voltage and frequency scaling, in: Proceedings of the 30th International Conference on Computers and Their Applications (CATA'15), 2015, pp. 63–70.

[36] H. Yu, B. Veeravalli, Y. Ha, S. Luo, Dynamic scheduling of imprecise-computation tasks on real-time embedded multiprocessors, in: Proceedings of the 2013 IEEE 16th International Conference on Computational Science and Engineering (CSE'13), 2013, pp. 770–777, http://dx.doi.org/10.1109/CSE.2013.118.

[37] C.C. Lin, Y.C. Syu, C.J. Chang, J.J. Wu, P. Liu, P.W. Cheng, W.T. Hsu, Energy-efficient task scheduling for multi-core platforms with per-core DVFS, J. Parallel Distrib. Comput. 86 (2015) 71–81, http://dx.doi.org/10.1016/j.jpdc.2015.08.004.

[38] G.L. Stavrinides, H.D. Karatza, A cost-effective and QoS-aware approach to scheduling real-time workflow applications in PaaS and SaaS clouds, in: Proceedings of the 3rd International Conference on Future Internet of Things and Cloud (FiCloud'15), 2015, pp. 231–239, http://dx.doi.org/10.1109/FiCloud.2015.93.

[39] W. Voorsluys, J. Broberg, S. Venugopal, R. Buyya, Cost of virtual machine live migration in clouds: a performance evaluation, in: Proceedings of the 1st International Conference on Cloud Computing (CloudCom'09), 2009, pp. 254–265, http://dx.doi.org/10.1007/978-3-642-10665-1_23.

[40] G.C. Buttazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, third ed., Springer, 2011, http://dx.doi.org/10.1007/978-1-4614-0676-1.

[41] G.L. Stavrinides, H.D. Karatza, The effect of workload computational demand variability on the performance of a SaaS cloud with a multi-tier SLA, in: Proceedings of the IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud'17), 2017, pp. 10–17, http://dx.doi.org/10.1109/FiCloud.2017.26.

[42] J.A. Butts, G.S. Sohi, A static power model for architects, in: Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'00), 2000, pp. 191–201, http://dx.doi.org/10.1109/MICRO.2000.898070.

[43] F. Diniz Rossi, M. Storch, I. de Oliveira, C.A.F. De Rose, Modeling power consumption for DVFS policies, in: Proceedings of the 2015 IEEE International Symposium on Circuits and Systems (ISCAS'15), 2015, pp. 1879–1882, http://dx.doi.org/10.1109/ISCAS.2015.7169024.

[44] K.H. Kim, R. Buyya, J. Kim, Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters, in: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID'07), 2007, pp. 541–548, http://dx.doi.org/10.1109/CCGRID.2007.85.

[45] G.L. Stavrinides, H.D. Karatza, Simulation-based performance evaluation of an energy-aware heuristic for the scheduling of HPC applications in large-scale distributed systems, in: Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering (ICPE'17), 3rd International Workshop on Energy-aware Simulation (ENERGY-SIM'17), 2017, pp. 49–54, http://dx.doi.org/10.1145/3053600.3053611.

[46] M. Park, S. Han, H. Kim, S. Cho, Y. Cho, Comparison of tie-breaking policies for real-time scheduling on multiprocessor, in: Proceedings of the 2004 International Conference on Embedded and Ubiquitous Computing (EUC'04), 2004, pp. 174–182, http://dx.doi.org/10.1007/978-3-540-30121-9_17.

[47] V. Sundriyal, M. Sosonkina, Z. Zhang, Automatic runtime frequency-scaling system for energy savings in parallel applications, J. Supercomput. 68 (2) (2014) 777–797, http://dx.doi.org/10.1007/s11227-013-1062-0.

[48] L. Brochard, F. Thomas, Energy aware scheduling, in: Proceedings of the 2013 Equip@Meso Computational Fluid Dynamics Symposium (CFD'13), 2013.

[49] Intel, https://ark.intel.com/products/ (Accessed: 26.03.18).

[50] Amazon Web Services - Amazon EC2 On-Demand Instance Pricing, https://aws.amazon.com/ec2/pricing/on-demand/ (Accessed: 09.10.18).

Georgios L. Stavrinides received the B.Sc. degree in Informatics from Aristotle University of Thessaloniki, Greece in 2006 and the M.Sc. degree in Advanced Computing from Imperial College London, UK in 2007. He received the Ph.D. degree in Computer Science from Aristotle University of Thessaloniki, Greece in 2014. He is currently a postdoctoral researcher in the Department of Informatics at the Aristotle University of Thessaloniki, Greece, under the supervision of Professor Emeritus Helen D. Karatza. His research interests include: scheduling algorithms, real-time distributed systems, fog and cloud computing, modeling, simulation and performance evaluation of large-scale distributed systems.

Helen D. Karatza is a Professor Emeritus in the Department of Informatics at the Aristotle University of Thessaloniki, Greece. Her research interests include: computer systems modeling and simulation, performance evaluation, grid and cloud computing, energy efficiency in large-scale distributed systems, real-time distributed systems, resource allocation and scheduling. She is the Editor-in-Chief of the Elsevier journal "Simulation Modelling Practice and Theory" and Senior Associate Editor of the "Journal of Systems and Software" of Elsevier. She has been Guest Editor of special issues in multiple international journals.