Journal of Systems Architecture 84 (2018) 12–27

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Soft error-aware energy-efficient task scheduling for workflow applications in DVFS-enabled cloud☆

Tingming Wu[a], Haifeng Gu[a], Junlong Zhou[b], Tongquan Wei[a], Xiao Liu[c], Mingsong Chen*,[a]

[a] Shanghai Key Lab of Trustworthy Computing, East China Normal University, Shanghai 200062, China
[b] School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China
[c] School of Information Technology, Deakin University, Melbourne 3125, Australia

ARTICLE INFO

Keywords:
Workflow
Cloud computing
Task scheduling
Soft error
Dynamic Voltage and Frequency Scaling
Reliability

ABSTRACT

Dynamic Voltage and Frequency Scaling (DVFS) has been widely used as a promising power management method to reduce the energy consumption of cloud workflows. However, due to the increasing chip density, lowering CPU voltages improperly in cloud data centers may inevitably increase soft error rate during workflow execution. Consequently, failures of timely completion of workflow applications may often take place, which raises serious concerns during the operation and maintenance of cloud data centers. To address such a problem, this paper proposes a soft error-aware energy-efficient task scheduling approach for workflow applications in DVFS-enabled cloud data centers. Under reliability and completion time constraints requested by tenants, our approach can generate energy-efficient task schedules for workflows by allocating tasks to appropriate virtual machines with specific operating frequencies. Comprehensive experiments on various well-known scientific workflow benchmarks show the effectivenss of our approach. Compared with state-of-the-art methods, our approach can achieve more than 30% energy savings while satisfying both reliability and completion time requirements.

1. Introduction

Along with the increasing popularity of pay-as-you-go cloud services, more and more enterprises and communities choose cloud computing platforms to deploy their commercial or scientific workflow applications, which facilitates the automated data distribution and computation-intensive applications [1]. To achieve increasing profit in the fierce cloud computing market, energy consumption is becoming a major concern of workflow service providers, as the rising energy consumption significantly contributes high operating costs and carbon footprints to the environment. Although the execution of more cloud workflow applications needs more energy, the high energy consumption of cloud workflow execution is mainly due to the inefficient utilization of computing resources [2].

As one of the most effective power management techniques, Dynamic Voltage and Frequency Scaling (DVFS) has been widely investigated to reduce energy consumption in cloud data centers [3,4]. By dynamically scaling down the processor frequency and voltage, DVFS enables a task to be accomplished before its deadline with the minimum energy consumption. Fig. 1 shows the basic components (i.e., workflow analyzer and task scheduler) and procedures of a DVFS-enabled scheduler adopted by most cloud workflow service providers. To carry out a workflow application, a user needs to submit the corresponding workflow request together with requirements of Quality of Service (QoS) to some cloud workflow service provider. Such requirements usually contain the information of expected completion time, reliability (bearable failure ratio), and so on. Once receiving a workflow request, the workflow analyzer will firstly analyze both the expected execution time of each task and precedence relations among tasks, and then send such information to the task scheduler. To efficiently execute workflow tasks based on cloud infrastructures, a typical workflow service provider usually maintains a large set of Virtual Machines (VMs). Based on the derived workflow task information and available VM resources, the task scheduler can figure out an energy-efficient schedule, where each workflow task is allocated to a VM with a specific frequency. Such a schedule can ensure minimized energy consumption without adversely affecting required QoS levels.

By properly lowering the frequency and voltage of VMs, the energy consumption of DVFS-enabled cloud workflow applications can be drastically reduced. However, since scaling down processor frequency

☆ This work was partially supported by Natural Science Foundation of China (Grant Nos. 61672230, 61300042, 61272420), Shanghai Municipal Natural Science Foundation (Grant No. 16ZR1409000), and ECNU Outstanding Doctoral Dissertation Cultivation Plan of Action (Grant No. PY2015047).

* Corresponding author.
E-mail address: mschen@sei.ecnu.edu.cn (M. Chen).

https://doi.org/10.1016/j.sysarc.2018.03.001
Received 5 November 2017; Received in revised form 9 January 2018; Accepted 1 March 2018
Available online 06 March 2018
1383-7621/

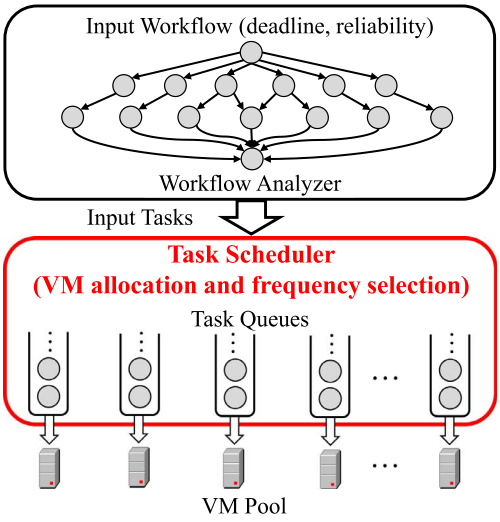


Fig. 1. Architecture of a DVFS-enabled cloud workflow scheduler.

and voltage increases the arrival rate of transient faults (i.e., soft errors), the reliability of workflow execution will be decreased [5]. This situation will become even worse since modern processors used in cloud data centers are based on CMOS technology. Typically, a CMOS processor contains billions of transistors where one or more transistors are used to represent a logic bit with the binary values of either 0 or 1. Unfortunately, various phenomena such as high energy cosmic particles and cosmic rays can cause the change of binary values held by transistors by mistake, resulting in the notorious soft errors. As the increasing number of transistors have a direct impact on the reliability of a chip [6], the susceptibility of cloud data centers to transient faults continuously increases [7]. In other words, the probability of incorrect results or system crashes due to soft errors during cloud workflow executions becomes increasingly higher.

To mitigate the impact of soft errors, checkpointing with rollback-recovery has been widely employed in cloud systems to ensure the system reliability [8]. By using this approach, the correct VM execution states are periodically saved in stable storages at each checkpoint. Whenever a soft error is detected, the rollback-recovery operation enables the restoration of the latest correct system state. Although checkpointing with rollback-recovery is promising, frequent utilization of such fault-tolerance operations is very time-consuming and expensive. The unpredictable overheads of checkpointing with rollback-recovery operations may inevitably cause severe temporal violations in cloud workflows [1]. Consequently, the timeliness of workflow execution results cannot be guaranteed. This is not acceptable in many scenarios, e.g., a daily weather forecast scientific workflow has to be accomplished before the broadcasting of the weather forecast programme.

To address the above problem, this paper proposes a novel three-phase approach that can generate energy-efficient and soft error resilient schedules for workflow applications considering both the checkpointing with rollback-recovery mechanism and DVFS-enabled cloud infrastructures. In the first phase, our approach leverages the well-known HEFT (Heterogeneous Earliest Finish Time) method [9] to generate a feasible schedule under the given completion time and reliability requirements by binding each task-VM pair with an appropriate operating frequency. Based on our developed task remapping approach, the second phase updates the task-to-VM mapping obtained from the first phase by moving tasks to corresponding VMs with higher energy-efficiency. To fully exploit the potential of spare time slots, a reliability-aware DVFS algorithm is devised in the last phase to further reduce the overall energy consumption. Under the given completion time and reliability constraints, experimental results show that our approach can generate a schedule with optimized energy consumption.

The rest of this paper is organized as follows. Section 2 presents related work on energy and reliability-aware scheduling for workflows. After an introduction to the modeling and problem definition of workflow scheduling in DVFS-enabled cloud in Section 3, Section 4 describes our approach in detail. Section 5 presents the experimental results. Finally, Section 6 concludes the paper.

2. Related work

Due to the fierce competition in todays cloud workflow market, to save the operating costs while guaranteeing quality of service, considerable scheduling algorithms have been designed to improve workflow energy consumption and completion time. For example, Topcuouglu et al. [9] introduced a Heterogeneous Earliest Finish Time (HEFT) method, which can schedule precedence constrained tasks for workflows on a set of heterogeneous processors with the objective of minimizing workflow completion time. To maintain sustainable cloud data centers with ever-increasing size, Cao and Zhu [10] proposed an energy-efficient workflow scheduling algorithm to minimize the energy consumption while satisfying a certain QoS level. Durillo et al. [11] developed a Pareto-based multi-objective algorithm for scheduling workflows, which can derive optimal solutions in terms of makespan and energy efficiency. Although the above works can effectively minimize the completion time or energy consumption of workflow applications, none of them considered reliability during the task scheduling.

To guarantee the reliability of embedded and cloud systems, various scheduling-based methods have been investigated. For example, Zhu et al. [12] focused on the slack allocation problem for a set of real-time tasks to minimize their energy consumption while preserving the overall system reliability. They identified that the problem is NP-hard and proposed two heuristic schemes to find optimized solutions. Wang et al. [13] proposed a look-ahead genetic algorithm that can optimize both the makespan and reliability of a workflow simultaneously. Rather than dealing with system-level reliability, Yan et al. [14] introduced a task-level reliability computation method based on the evaluation of dynamic service capacity of resources. By wisely managing resources, they proposed a reliability enhanced workflow scheduling algorithm to maximize the reliability of task executions under budget constraint. Zhang et al. [15] developed a bi-objective genetic algorithm (BOGA) to pursue low energy consumption and high system reliability for workflow scheduling. As two mainstream fault-tolerance techniques, execution replication and checkpointing are promising to enhance the reliability of cloud data centers. For example, Sedaghat et al. [16] presented an approximation technique based on their statistical reliability models, which can compute the reliability of task execution in cloud data centers considering correlated failures. To satisfy reliability constraints, they proposed an effective scheduling algorithm that can achieve the desired reliability with a minimum number of replicas. To construct soft error resilient cloud computing systems, Gao et al. [17] proposed an energy-aware fault-tolerant scheduling framework. By using error detection and active replication, the framework can achieve high error coverage and fault tolerance confidence. Based on checkpointing techniques, Zhou et al. [18] proposed a heuristic that can not only guarantee service reliability, but also minimize network and storage resource usage in cloud. Although the above approaches can guarantee the reliability of cloud workflows, few of them consider the impact of DVFS on transient fault occurrences.

As an efficient power management technique, DVFS has been widely used in reducing the energy consumption of cloud computing systems. For example, Wang et al. [3] presented an energy-efficient method to schedule precedence-constrained parallel tasks in a DVFS-enabled cluster by efficiently utilizing slack time for non-critical jobs. Jiang et al. [19] presented a DVFS-based scheduling approach, which can make a reasonable trade-off between execution time and energy consumption for workflow applications. Huang et al. [20] proposed an Enhanced Energy-efficient Scheduling (EES) algorithm for parallel applications to reduce energy consumption while meeting the

performance-based service level agreement (SLA). Lin et al. [4] proposed a novel DVFS-based algorithm for mobile cloud computing. It can satisfy the specified task-precedence requirements and the application completion time constraint while minimizing the total energy dissipation in a mobile device. Tang et al. [21] proposed a DVFS-enabled Energy-efficient Workflow Task Scheduling (DEWTS) algorithm, which can optimize the energy savings while meeting the deadlines for parallel applications. Our approach is inspired by both EES and DEWTS. We also utilize the HEFT algorithm and DVFS to optimize the overall energy consumption of workflow tasks. However, unlike the above methods, only our approach considers the reliability issues.

To mitigate the overheads incurred by checkpointing with rollback recovery technique, Salehi et al. [22] introduced a two-state checkpointing scheme for fault-tolerant hard real-time systems. By combing with dynamic voltage scaling technique, their approach can achieve significant energy saving while satisfying reliability constraints. However, their approach only considers independent application tasks without considering the precedence relations among tasks. Similar to our work, Zheng et al. [8] presented a strategy that utilizes DVFS and checkpointing techniques to minimize energy consumption for workflow execution while meeting the requirements of reliability and deadline. However, their work focuses on the design of checkpointing and recovery strategies rather than task scheduling. To the best of our knowledge, our work is the first attempt to minimize energy consumption of cloud workflows while considering soft errors in DVFS-enabled cloud data centers.

3. Models and problem formulation

The goal of this paper is to obtain energy-efficient task schedules for given workflows while guaranteeing tenant-defined reliability and completion time constraints within DVFS-enabled cloud data centers. This section presents the system models and defines the scheduling problem. To explain our problem more clearly, Table 1 summarizes all the notations used in our approach.

3.1. Modeling of virtual machines

Consider a cloud platform that provides a set of heterogeneous VMs denoted by $S = \{s_1, s_2, ..., s_n\}$. Each VM is assumed to execute on a DVFS-enabled physical processor that can operate at different voltage and frequency levels. The physical processor of VM s_i is characterized by a triplet (V_i, F_i, c_i), where $V_i = \{v_{i,1}, v_{i,2}, ..., v_{i,k}\}$ represents the voltage set, $F_i = \{f_{i,1}, f_{i,2}, ..., f_{i,k}\}$ indicates the clock frequency set, and c_i denotes the maximum processing capacity of the processor assigned to s_i. To simplify modeling, we have $0 < v_i^{\min} = v_{i,1} < v_{i,2} < ... < v_{i,k} = v_i^{\max}$ and $0 < f_i^{\min} = f_{i,1} < f_{i,2} < ... < f_{i,k} = f_i^{\max}$, where $v_i^{\min}$, $v_i^{\max}$, $f_i^{\min}$ and $f_i^{\max}$ indicate the minimum voltage, maximum voltage, minimum frequency and maximum frequency of the processor assigned to VM s_i, respectively. When a processor operates at a clock frequency of $f_{i,j}$, its corresponding supply voltage will be $v_{i,j}$. The processing capacity of VM s_i is measured in Million Instructions Per Second (MIPS), which is proportional to its specified clock frequency. In our model, we allow VMs to communicate with each other. We use $b_{i,j}$ to indicate the network bandwidth between VM s_i and VM s_j. Similar to the work presented in [23], we do not consider network congestion in this paper.

3.2. Modeling of workflows

The structure of a workflow W can be modeled as a Directed Acyclic Graph (DAG) $G = (\mathscr{T}, E)$, where $\mathscr{T} = \{\tau_1, \tau_2, ..., \tau_m\}$ represents the task set and E represents the edge set of the graph. Each edge $e_{uv} \in E$ in the form of (τ_u, τ_v) indicates that there is a precedence constraint between task τ_u and task τ_v, where task τ_u is an immediate predecessor of task τ_v and task τ_v is an immediate successor of task τ_u. Since a task may have multiple predecessors and successors, we use $pred(\tau_u)$ and $succ(\tau_u)$ to denote the set of immediate predecessors and successors of task τ_u, respectively. A task without any predecessors is called an entry task, and a task without any successors is called an exit task.

To model the communication between tasks, each edge e_{uv} is associated with a weight $w_{u,v}$, which denotes the size of data that needs to be transferred from task τ_u to task τ_v. Note that a task cannot start execution until it completely receives the output data from all of its predecessors. Assume that task τ_u and task τ_v are assigned to VM s_i and VM s_j, respectively. If $i \neq j$, the communication cost between these two tasks can be calculated as $\frac{w_{u,v}}{b_{i,j}}$. Otherwise, since both tasks share the same VM (i.e., $i = j$), the communication cost will be zero.

3.3. Modeling of tasks with fault recovery

In our approach, a workflow task is the basic execution unit. A VM can only execute one task at a time, and a task cannot change its specified processor frequency when running on a VM. Since our model supports DVFS, a frequency transition of VM may occur when a new task starts. Let c_i be the processing capacity (in MIPS) of VM s_i operating at its maximum clock frequency $f_i^{\max}$, i_u be the number (in millions) of instructions that task τ_u needs to execute, and $f_i(\tau_u)$ be the actual running clock frequency of task τ_u on VM s_i. The execution time of task τ_u on VM s_i at the maximum clock frequency $f_i^{\max}$ is $\frac{i_u}{c_i}$. When the clock frequency of VM s_i is set to $f_i(\tau_u)$, the execution time of task τ_u can be calculated as $\frac{i_u}{c_i} \cdot \frac{f_i^{\max}}{f_i(\tau_u)}$.

Due to the ever-increasing trend of technology scaling and wide use of DVFS technique, CMOS logic circuits are becoming more vulnerable to transient faults. To ensure the required system reliability, in this paper we adopt equidistant checkpointing technique [22,24], which assumes that the lengths of checkpoint intervals for a given task are the same. At each checkpoint of a running task, the latest correct task state is saved in a secure device to enable rolling back to the most recent checkpoint by restoring its state to the saved value. In case of no fault occurrences, we can get the best case execution time of a task that can be calculated as the sum of original task execution time and checkpoint overhead. Let $N_{i,u}(f)$ denote the number of checkpoints deployed during the execution of task τ_u on VM s_i with a frequency of f. The best case execution time $\Phi_{i,u}^{best}$ of task τ_u on VM s_i with a given frequency $f_i(\tau_u)$ can be calculated using

$$\Phi_{i,u}^{best}(f_i(\tau_u)) = \frac{i_u}{c_i} \cdot \frac{f_i^{max}}{f_i(\tau_u)} + N_{i,u}(f_i(\tau_u)) \cdot O_{cp}, \tag{1}$$

where O_{cp} denotes the time overhead of checkpointing.

Whenever a transient fault occurs during the execution of a task, the most recent checkpoint saved in the secure device will be used to recover task execution. Let

$$l_{i,u}(f_i(\tau_u)) = \frac{1}{N_{i,u}(f_i(\tau_u)) + 1} \cdot \frac{i_u}{c_i} \cdot \frac{f_i^{max}}{f_i(\tau_u)} \tag{2}$$

denote the checkpoint interval length of task τ_u running on VM s_i at a frequency of $f_i(\tau_u)$. Let K_u denote the worst case number of fault occurrences during the execution of task τ_u. The worst case execution time $\Phi_{i,u}$ of task τ_u on VM s_i equals the sum of the best case execution time $\Phi_{i,u}^{best}$ and the fault recovery overhead in the presence of the maximum K_u fault occurrences. Thus, $\Phi_{i,u}$ is formulated as

$$\Phi_{i,u}(f_i(\tau_u)) = \Phi_{i,u}^{best}(f_i(\tau_u)) + K_u \cdot l_{i,u}(f_i(\tau_u)) + 2 \cdot K_u \cdot O_{cp}, \tag{3}$$

where $K_u \cdot l_{i,u}(f_i(\tau_u))$ indicates the recovery overhead of tolerating K_u faults, and $2 \cdot K_u \cdot O_{cp}$ represents the accumulative overhead of K_u checkpoint saving operations and K_u checkpoint retrieval operations [25]. Note that in our model the fault detection and fault recovery are two separate processes. The fault detection process is triggered upon the finish of a checkpoint interval. Similar to the checkpointing mechanism presented in [26], we assume that the fault detection overhead

Table 1
Definitions of notations used to solve our problem.

Notations	Definition
$S = \{s_1, s_2, \cdots, s_n\}$	A set of VMs.
$\mathscr{T} = \{\tau_1, \tau_2, \cdots, \tau_m\}$	A task set of the given workflow.
$F_i = \{f_{i,1}, f_{i,2}, \cdots, f_{i,k}\}$	The supply frequency set of VM s_i.
$f_i^{\max}$, $f_i^{\min}$	The maximum and minimum supply frequencies of VM s_i.
$pred(\tau_u)$, $succ(\tau_u)$	The set of immediate predecessors and successors of task τ_u in the given workflow.
$f_i(\tau_u)$	The actual running clock frequency of task τ_u on VM s_i.
$w_{u,v}$	The size of data that needs to be transferred from task τ_u to τ_v.
$b_{i,j}$	The network bandwidth between VM s_i and s_j.
i_u	The number (in millions) of instructions of task τ_u.
c_i	The processing capacity of VM s_i operating at a frequency of $f_i^{\max}$.
O_{cp}	The time overhead of one checkpoint.
$l_{i,u}(f)$	The checkpoint interval length of task τ_u running on VM s_i with a frequency of f.
K_u	The maximum number of fault occurrences of task τ_u.
$\Phi_{i,u}^{best}(f)$, $\Phi_{i,u}(f)$	The best and worst case execution time of task τ_u on VM s_i with a frequency of f.
$N_{i,u}(f)$	The number of checkpoints deployed during the execution of task τ_u on VM s_i with a frequency of f.
$N_{i,u}^{opt}(f)$	The optimal number of checkpoints that minimizes $\Phi_{i,u}(f)$.
$\lambda_i(f)$	The average arrival rate of soft errors on VM s_i at the frequency of f.
$\lambda_{i,0}(f)$	The average arrival rate of soft errors on VM s_i at the maximum frequency $f_i^{\max}$.
ξ_i	A system dependent parameter of VM s_i that measures the sensitivity of the fault arrival rate to frequency scaling.
$P_{i,u}(k,f)$	The probability of k faults occurring during the execution of task τ_u that runs on VM s_i at a frequency of f.
$R_{i,u}(k,f)$	The probability of successfully recovering k faults when task τ_u runs on VM s_i at a frequency of f.
$R_{i,u}(f)$	The reliability of task τ_u on VM s_i with a frequency of f.
$P_{i,u}(f)$	The power consumption of task τ_u on VM s_i with a frequency of f.
$E_{i,u}(f)$	The energy consumption of task τ_u on VM s_i with a frequency of f.
$ST_{i,u}(f)$, $FT_{i,u}(f)$	The actual start and finish time of task τ_u on VM s_i with a frequency of f.
A	A task-to-VM mapping for a task schedule, where $A_u = i$ holds if task τ_u is allocated to VM s_i.
FL^A	The frequency level assignment of all tasks on their host VMs based on A, where $FL_{i,u}^A = l$ indicates $f_i(\tau_u) = f_{i,l}$.
$R(A, FL^A)$	The workflow reliability based on A and FL^A.
$E(A, FL^A)$	The total energy consumption of a workflow based on A and FL^A.
$M(A, FL^A)$	The completion time of a workflow based on A and FL^A.
R_{goal}	The required reliability for a given workflow.
D	The required completion time for a given workflow.

is negligible.

Let $N_{i,u}^{opt}(f_i(\tau_u))$ denote the optimal number of checkpoints that minimizes the worst case execution time $\Phi_{i,u}(f_i(\tau_u))$ of task τ_u on VM s_i at the specified frequency, i.e., $f_i(\tau_u)$. According to the inequality of arithmetic and geometric means, $N_{i,u}^{opt}(f_i(\tau_u))$ can be derived from Eq. (3). It can be formulated as

$$N_{i,u}^{opt}(f_i(\tau_u)) = \left\lfloor \sqrt{\frac{K_u}{O_{cp}} \cdot \left(\frac{i_u}{c_i} \cdot \frac{f_i^{max}}{f_i(\tau_u)} \right)} - 1 \right\rfloor. \tag{4}$$

We can find that $N_{i,u}^{opt}$ is independent from the arrival rate of transient faults. This is because in this formula we only consider the worst case number of fault occurrences (i.e., K_u). Note that to simplify the model, in general case we assume that $N_{i,u}(f_i(\tau_u))$ equals $N_{i,u}^{opt}(f_i(\tau_u))$.

Assuming that the average arrival rate of transient faults on VM s_i at the frequency of f is $\lambda_i(f)$, which can be calculated using $\lambda_i(f) = \lambda_{i,0} \cdot 10^{\xi_i \cdot \frac{f_i^{\max} - f}{f_i^{\max} - f_i^{\min}}}$ [5], where $\lambda_{i,0}$ represents the average fault arrival rate when VM s_i operates at the maximum frequency $f_i^{\max}$, and ξ_i is a system-dependent parameter that measures the sensitivity of the fault arrival rate to frequency scaling. Typically, the occurrence pattern of transient faults follows the Poisson distribution. Therefore, the probability of k transient faults occurring during the execution of task τ_u on VM s_i at the frequency of $f_i(\tau_u)$ can be formulated as

$$P_{i,u}(k, f_i(\tau_u)) = \frac{e^{-\lambda_i(f_i(\tau_u)) \cdot \Phi_{i,u}(f_i(\tau_u))} \cdot (\lambda_i(f_i(\tau_u)) \cdot \Phi_{i,u}(f_i(\tau_u)))^k}{k!}. \tag{5}$$

The reliability of a task is defined as the probability that the task can successfully finish all its execution in the presence of soft errors. In the worst case of k transient fault occurrences, to ensure the successful completion of task τ_u on VM s_i, k task segments in between consecutive checkpoints are re-executed to tolerate these faults. Since the probability of successfully recovering k faults is $e^{\lambda_i(f_i(\tau_u)) \cdot (k \cdot l_{i,u})}$ [8], the probability of successfully executing task τ_u can be calculated using

$$R_{i,u}(k, f_i(\tau_u)) = P_{i,u}(k, f_i(\tau_u)) \cdot e^{\lambda_i(f_i(\tau_u)) \cdot (k \cdot l_{i,u})}. \tag{6}$$

Since K_u indicates the maximum number of fault occurrences, a task τ_u is considered to be reliable if it can tolerate up to K_u errors during its execution. Otherwise, the task τ_u will fail to finish if there are more than K_u transient fault occurrences during its execution. Therefore, the reliability $R_{i,u}$ of task τ_u on VM s_i at the frequency of $f_i(\tau_u)$ can be formulated as

$$R_{i,u}(f_i(\tau_u)) = \sum_{k=0}^{K_u} P_{i,u}(k, f_i(\tau_u)) \cdot e^{\lambda_{i,0} \cdot (k \cdot l_{i,u})}. \tag{7}$$

3.4. Modeling of energy consumption

Generally speaking, energy consumption of data centers comes from various sources including cooling systems, network equipments and physical processors. To simplify the modeling, in this paper we only consider the energy consumption of physical processors employed by VMs. This is mainly due to the two following reasons. First, processors dominate system-wide power consumption [27]. Reducing the average power consumed by physical processors can lead to lower overall energy consumption of VMs. Second, based on the observations by Guerout et al. [28], the system-wide power consumption is in proportion to the power consumed by processors. Therefore, if the energy consumption of physical processors can be decreased by task scheduling, the system-wide energy consumption can also be reduced accordingly.

During the execution of a VM, its power consumption consists of two parts, i.e., dynamic power and static power. The power

consumption of executing task τ_u on VM s_i can be formulated as

$$P_{i,u}(f_i(\tau_u)) = P^d_{i,u}(f_i(\tau_u)) + P^s_{i,u}, \tag{8}$$

where $P^d_{i,u}$ and $P^s_{i,u}$ denote the dynamic power and static power of executing task τ_u on VM s_i, respectively. Since each VM executes on a DVFS-enabled processor, the dynamic power $P^d_{i,u}$ of VM s_i can be expressed as a function of its given supply voltage $v_i(\tau_u)$ and operating frequency $f_i(\tau_u)$ [27,29], i.e.,

$$P^d_{i,u}(f_i(\tau_u)) = \alpha_i \cdot v_i^2(\tau_u) \cdot f_i(\tau_u), \tag{9}$$

where α_i is a processor related constant. To evaluate the energy consumption of a task, we investigate the worst case total energy $E_{i,\ u}$ of task τ_u running on VM s_i at the frequency $f_i(\tau_u)$. It can be calculated as follows:

$$E_{i,u}(f_i(\tau_u)) = P_{i,u}(f_i(\tau_u)) \cdot \Phi_{i,u}(f_i(\tau_u)). \tag{10}$$

3.5. Problem formulation

The objective of this paper is to find a task schedule that minimizes the energy consumed by tasks of a given workflow under the reliability and completion time constraints. This is mainly because that completion time and reliability are two major concerns of cloud tenants, which need to be clearly specified in service level agreements. Let $A: \mathscr{T} \rightarrow S$ denote a task-to-VM mapping in a task schedule. We use A_u to indicate the index of the VM assigned to task τ_u ($1 \le u \le m$). For each task τ_u in the workflow, $A_u = i$ holds if it is allocated to VM s_i in A. Let FL^A be the frequency level assignment of all the tasks on their host VMs based on A, where $FL^A_{i,u} = l$ indicates that task τ_u is assigned to VM s_i with a specified frequency, i.e., $f_i(\tau_u) = f_{i,l}$. Note that if $A_u \neq i$, $FL^A_{i,u}$ is set to -1 which indicates that τ_u cannot be allocated to s_i. In this paper, a task schedule is defined as a set of triplets in the form of $\left\{\left(\tau_u, s_{A_u}, f_{A_u,FL^A_{A_u,u}}\right) \middle| 1 \le u \le m\right\}$, where s_{A_u} denotes the VM assigned to τ_u and $f_{A_u,FL^A_{A_u,u}}$ denotes the specified frequency when executing τ_u on s_{A_u}. Note that a task schedule can be derived from the pair of A and FL^A.

Since a workflow consists of a set of tasks with precedence constraints, a workflow is considered to be finished successfully when all its tasks are finished successfully. Therefore, based on the given A and FL^A for workflow W, we can calculate the workflow reliability R as the cumulative product of reliabilities of all the workflow tasks [15,30]. It is formulated as

$$R(A, FL^A) = \prod_{u=1}^{m} R_{A_u,u}\left(f_{A_u,FL^A_{A_u,u}}\right). \tag{11}$$

Assuming that the workflow has a reliability constraint of R_{goal}, the derived task schedule satisfies the reliability requirement only when $R_{goal} \le R$ holds.

Given a workflow, we only consider the energy consumed by all its tasks. We do not consider the energy consumed by VMs in their idle states, or the energy wasted by turning on/off VMs, though the leakage current or static power has dominated in the overall energy consumption when the size of transistor is shrinking. This is mainly because more and more servers support sleep mode, where the servers switch off unneeded subsystems and put the RAM into a minimum power state. In the sleep mode, the servers have a low power consumption. For example, IBM is developing a new *deep-sleep* mode for its Power processors that will allow them to consume almost no power when they are idle. Moreover, the overhead of switching modern processors from sleep mode to active mode is only about 300 ms [31]. Comparing with the long execution time of workflow tasks, such on/off switching overhead of VMs is negligible. Based on the energy consumption model for an individual task presented in Eq. (10), the energy consumption of a workflow can be formulated as

$$E(A, FL^A) = \sum_{u=1}^{m} E_{A_u,u}\left(f_{A_u,FL^A_{A_u,u}}\right). \tag{12}$$

Let $ST_{i,\ v}$ and $FT_{i,\ v}$ denote the actual start time and actual finish time of task τ_v on VM s_i following a task schedule, respectively. According to the precedence constraints, $ST_{i,\ v}$ can be calculated using

$$ST_{i,v}(f_i(\tau_v)) = \begin{cases} 0 & \tau_u \text{ is an entry task} \\ \max\limits_{\tau_u \in pred(\tau_v)} \left\{ FT_{A_u,u}\left(f_{A_u,FL^A_{A_u,u}}\right) + \frac{w_{u,v}}{b_{A_u,A_v}} \right\} & A_u \neq A_v \\ \max\limits_{\tau_u \in pred(\tau_v)} \left\{ FT_{A_u,u}\left(f_{A_u,FL^A_{A_u,u}}\right) \right\} & \text{otherwise} \end{cases} \tag{13}$$

The actual finish time of task τ_v can be approximated as $FT_{i,v}(f_i(\tau_v)) = ST_{i,v}(f_i(\tau_v)) + \Phi_{i,v}(f_i(\tau_v))$ considering fault recovery overheads. Therefore, the workflow makespan M can be calculated as

$$M(A, FL^A) = \max_{\tau_u \in \mathscr{T}} \left\{ FT_{A_u,u}\left(f_{A_u,FL^A_{A_u,u}}\right) \right\}. \tag{14}$$

Consider a workflow W consisting of a set of tasks $\mathscr{T}$ with precedence constraints E, and a set of heterogeneous VMs S supporting different voltage and frequency levels. This paper tries to find a task-to-VM mapping A and corresponding frequency level assignment FL^A, such that the energy consumption $E(A, FL^A)$ of the workflow can be optimized without violating the specified workflow completion time requirement D and reliability constraint R_{goal}. The problem to be addressed can be formalized as follows:

$$\text{Minimize:} \quad E(A, FL^A) \tag{15}$$

$$\text{subject to:} \quad R_{goal} \le R(A, FL^A), \tag{16}$$

$$M(A, FL^A) \le D. \tag{17}$$

It is worth noting that although scaling down the frequencies of tasks can reduce the overall energy consumption of a workflow, due to the increasing chance of transient faults more checkpointing operations may be required. This may prolong the task execution time, resulting in completion time violation or even the increase of energy consumption. It has been proven that the problem of reliability-aware energy management for multiple tasks is NP-hard [12]. However, the problem introduced in [12] did not consider task dependencies. Since our task scheduling problem tries to optimize the allocation of multiple dependent tasks to different VMs with appropriate operating frequencies, it is more complex than the work presented in [12]. Therefore, the problem to be solved is NP-hard. To quickly obtain a solution, this paper proposes a three-phase heuristic which can find a near-optimal task schedule for a given workflow. The details of our approach are described in the following section.

4. Our task scheduling approach

Fig. 2 presents an overview of our task scheduling approach. The proposed approach consists of three phases: initial scheduling, task remapping and DVFS. Given an input workflow (a set of precedence constrained tasks), a set of heterogeneous VMs, a target completion time D and a target reliability R_{goal}, our approach can generate a task schedule which consists of a task-to-VM mapping A and a frequency level assignment FL^A. In the first phase, the initial scheduling algorithm firstly checks all the combinations of tasks, VMs and corresponding frequency levels, and then figures out one feasible task-to-VM mapping together with selected frequencies using the HEFT approach [9]. Since HEFT always chooses a powerful VM for each task to minimize task completion time rather than an energy-efficient VM, the energy consumption of the obtained schedule is generally not optimal. Therefore, in the second phase we develop a task remapping algorithm that can reduce the overall energy consumption by updating the task-to-VM

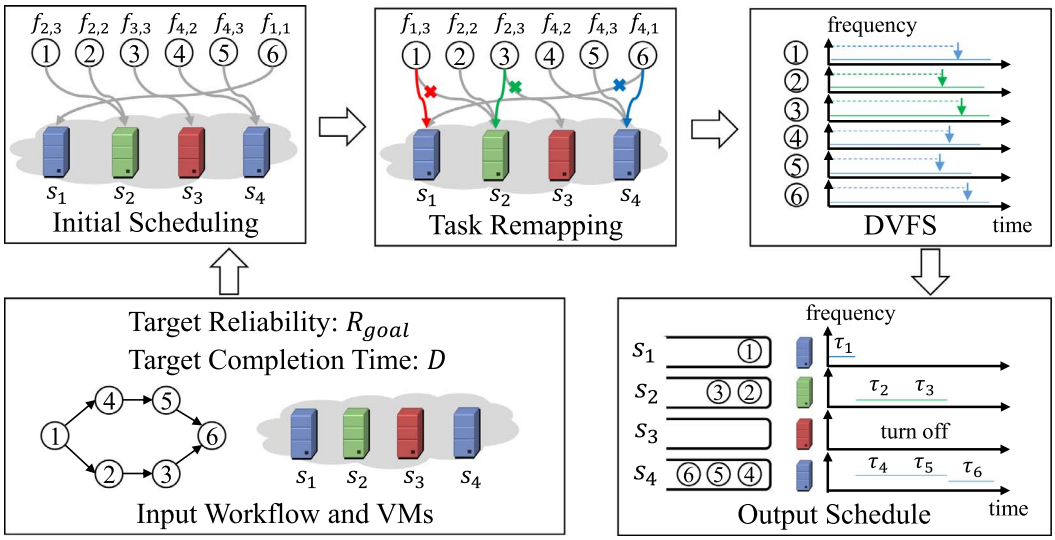


Fig. 2. An overview of our approach.

mapping obtained in the first phase. Moreover, due to the precedence constraints among tasks, spare time slots may exist between each pair of neighboring tasks on non-critical workflow paths. To fully take advantage of these spare time slots, a reliability-aware DVFS algorithm is devised in the third phase to further optimize the energy consumption. Finally, a task schedule with optimized energy consumption will be reported satisfying both completion time and reliability constraints. The following subsections will introduce each phase of our approach in detail.

4.1. Initial scheduling algorithm

HEFT is a promising heuristic that can generate a minimal-delay schedule for a given workflow on a set of heterogeneous VMs. When searching for a schedule, HEFT firstly ranks the workflow tasks based on their average execution cost, average communication cost and precedence constraints. Given the execution time $\Phi_{i,u}$ of task τ_u on VM s_i, the average computation cost Φ_u of task τ_u is calculated as the average execution time of τ_u on all the VMs, i.e.,

$$\Phi_u = \underset{1\le i\le n}{\text{avg}}\{\Phi_{i,u}\}. \tag{18}$$

Note that the value of $\Phi_{i,u}$ is strongly affected by the assigned frequency according to Eq. (3). Therefore, different frequency assignment FL^A of a task-to-VM mapping A will lead to different task ranks. Let $b_{i,j}$ denote network bandwidth between VM s_i and VM s_j. The average communication bandwidth b_{avg} among VMs is represented by

$$b_{avg} = \underset{1\le i\le n, 1\le j\le n}{\text{avg}}\{b_{i,j}\}. \tag{19}$$

Based on the average communication bandwidth b_{avg}, the average communication cost between task τ_u and τ_v can be calculated as

$$c_{u,v} = \frac{w_{u,v}}{b_{avg}}. \tag{20}$$

Based on the above definitions, the rank of task τ_u ($1 \le u \le m$) is defined as

$$rank_u = \begin{cases} \Phi_u & if\ \tau_u\ is\ an\ exit\ task \\ \Phi_u + \max\limits_{\tau_v \in succ(\tau_u)}\{c_{u,v} + rank_v\} & otherwise \end{cases}. \tag{21}$$

Note that the rank information implicitly reflects the precedence relations among tasks. It can be recursively calculated by traversing the workflow structure starting from exit tasks.

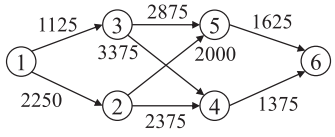
After calculating the ranks for all the tasks, HEFT starts to dispatch tasks to VMs one by one. At a time, HEFT selects an unassigned task with the highest rank and assigns it to an available VM that can minimize the task completion time. Once all the tasks are assigned, a task-to-VM mapping A is generated. Although original HEFT can quickly find a short schedule, it does not take the DVFS into account and its optimization objective is to minimize the completion time. To allow the tuning of task frequencies, the first phase of our approach extends the HEFT method. It enables the generation of a feasible schedule with low energy consumption while satisfying both reliability and completion time constraints.

Algorithm 1 presents the implementation details of our initial scheduling algorithm. The inputs of this algorithm are a set of VMs provided by a cloud data center, a given workflow submitted by some cloud tenant with a task set $\mathscr{T} = \{\tau_1, \tau_2, ..., \tau_m\}$, and workflow completion time and reliability constraints (i.e., D and R_{goal}).

In Algorithm 1, step 1 initializes two data structures FL and Θ, which are both two-dimensional arrays with a size of $n \times m$. We use FL to record the selected frequency levels for all the combinations of tasks and VMs, where $FL_{i,u}$ denote the selected frequency level for task τ_u on VM s_i. Note that FL is different from FL^A. FL^A is only a projection of FL based on the task-to-VM mapping A. We use Θ to record the worst execution time of all the task and VM combinations for the selected frequency levels. Steps 2-5 enumerate all the task-VM pairs and select a best frequency level for each pair with the minimum energy consumption. During the enumeration, if a better frequency with lower energy consumption is found, step 3 will update the lowest energy consumption for the pair of τ_u and s_i searched so far, and steps 4–5 will update the worst execution time $\Theta_{i,u}$ and frequency level $FL_{i,u}$ accordingly. Based on HEFT approach, step 6 figures out a preliminary task-to-VM mapping A for the workflow based on the obtained Θ information. Note that based on A and the selected frequency levels saved in FL, we can derive a task schedule for the workflow. Steps 7–8 calculates the reliability and makespan of the schedule based on A and FL^A. If the derived schedule violates either the reliability constraint R_{goal} or the completion time constraint D, steps 9–15 will try to iteratively adjust the frequency levels of all the task-VM pairs to achieve a better reliability and shorter completion time. To save the generation time of an initial near-optimal schedule, in the algorithm we do not check all the combinations of task-VM pairs and frequencies. Instead, we tune the frequency levels in a coarse manner as shown in steps 9–11. In each iteration of frequency adjustment, step 10 will increase one frequency level of the pair τ_u and s_i if its current frequency $f_{i,l}$ is not the highest, and step 11 will update the worst execution time for the pair based on the new frequency. After all the task-VM pairs updating their

Input: i) A set of VMs $S = \{s_1, s_2, \ldots, s_n\}$;
ii) A workflow W with task set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_m\}$;
iii) A given target completion time D;
iv) A given target reliability R_{goal};
Output: An initial task schedule
genInitSchedule(S, W, D, R_{goal})
begin
1. Initialize two-dimensional arrays FL and Θ with -1;
foreach τ_u *in* $W.\mathcal{T}$ **do**
foreach s_i *in* S **do**
2. $etemp = +\infty$;
foreach $f_{i,j}$ *in* $s_i.F_i$ **do**
if $E_{i,u}(f_{i,j}) < etemp$ **then**
3. $etemp = E_{i,u}(f_{i,j})$; // Equation(10)
4. $\Theta_{i,u} = \Phi_{i,u}(f_{i,j})$; //Equation (3)
5. $FL_{i,u} = j$;
end
end
end
end
6. $A = HEFT(S, W, \Theta)$;
7. $R = R(A, FL^A)$; // Equation (11)
8. $M = M(A, FL^A)$; // Equation (14)
while $R < R_{goal} \bigvee D < M$ **do**
foreach τ_u *in* $W.\mathcal{T}$ **do**
foreach s_i *in* S **do**
9. $l = FL_{i,u}$;
if $f_{i,l} < f_i^{\max}$ **then**
10. $l = l + 1$; $FL_{i,u} = l$;
11. $\Theta_{i,u} = \Phi_{i,u}(f_{i,l})$; // Equation (3)
end
end
end
12. $A = HEFT(S, W, \Theta)$;
13. $R = R(A, FL^A)$; // Equation (11)
14. $M = M(A, FL^A)$; // Equation (14)
end
15. return (A, FL);
end

Algorithm 1. Initial scheduling algorithm.



Tasks	1	2	3	4	5	6
Workload	18	18	18	8	20	15

Fig. 3. A workflow example and its task settings.

frequencies, step 12 will generate a new schedule and steps 13–14 will re-calculate its reliability and makespan. Finally, step 15 reports the obtained initial schedule in the form of (*A, FL*), where *A* represents the task-to-VM mapping and *FL* contains the selected frequencies for *A*. Note that during the frequency adjustment (steps 9–11) all the frequencies of task-VM pairs are increased monotonically. In this way, the occurrence chance of soft errors becomes less. Due to the increased frequency levels and reduced number of checkpoints, the worst case execution time and reliability of each task-VM pair will also be improved monotonically. Consequently, when more frequency adjustment iterations are executed, the generated initial schedule is more closer to a feasible schedule satisfying the given constraints.

Fig. 3 presents an example of a workflow with six tasks (i.e., $\tau_1 - \tau_6$). The DAG illustrates the structure of the workflow and the communication overhead (in Mb) between tasks. For example, when task τ_1 finishes, it will send a data block with a size of 1125 Mb to τ_3. The table in Fig. 3 shows the number of instructions (in billions) that the tasks need to execute. We assume that there are four VMs (i.e., $s_1 - s_4$), where VMs s_1 and s_2 execute on AMD Turion MT-34 processors, and other two VMs execute on AMD Opteron 2218 processors. The configurations of both processors can be found in Table 4 in Section 5.1. For example, task τ_1 has 18 billion instructions and the lowest frequency (i.e., $f_{3,1}$) of AMD Opteron 2218 processors is 1.0 GHz. Since each processor has two hardware threads, it needs 9 s to execute task τ_1 with the frequency of $f_{3,1}$. However, by considering the overhead of K_u soft errors, the worst case execution time (i.e., $\Phi_{i,u}(f)$) at the frequency of $f_{3,1}$ is 13.81 s. In this example, we set the network bandwidth between two tasks to 1 Gbps. For the workflow, we assume that its target reliability R_{goal} is 0.996 and its target completion time D is 48 s.

Our initial scheduling algorithm consists of two parts. The first part (steps 1–8) is to generate an intermediate schedule for the given workflow. Based on the most energy-efficient task-VM pair information (i.e., indicated by Θ in Algorithm 1), this part tries to obtain a schedule with optimized completion time using the HEFT algorithm. Table 2 shows the intermediate schedule generated from the workflow presented in Fig. 3. Note that in this example all the tasks are allocated to the VMs s_3 and s_4, because the HEFT algorithm does not take the energy consumption into account. Based on Eq. (11) and Eq. (14), we can get that the reliability and completion time of the immediate schedule are 0.9925 and 57.25 s, respectively. Neither of them satisfies the given workflow constraints. The overall energy consumption of the immediate schedule is 1803 J. The second part of our initial scheduling algorithm tries to adjust task frequencies to meet both the completion time and reliability constraints. Table 2(b) presents the schedule generated by the whole algorithm. We can find that by increasing the frequency levels of tasks, we can achieve a feasible schedule under the given completion time and reliability constraints. In this case, the schedule can be finished within 37.25 s with a reliability of 0.9995. The energy consumption of this schedule is 2271 J, which is more than the one of the immediate schedule.

4.2. Task remapping algorithm

In Algorithm 1, the initial schedule is mainly generated on top of the HEFT approach, which does not take DVFS into account. Once the frequency levels of task-VM pairs are selected, HEFT aims at figuring out one task-to-VM mapping with minimum completion time rather than the lowest energy consumption. Furthermore, the granularity of frequency adjustment is coarse in Algorithm 1. It does not consider all possible combinations of task-VM pairs and VM frequencies. Consequently, the initial schedule may not have the best task-to-VM mapping that can lead to the lowest energy consumption. Therefore, in the second phase our approach tries to optimize the initial schedule by tuning its task-to-VM mapping. The goal of our task remapping algorithm presented in Algorithm 2 is to search for a better task-to-VM mapping to further reduce the overall energy consumption of the given workflow.

Table 2
Results of initial scheduling algorithm.

(a) Intermediate schedule generated by steps 1–8 of Algorithm 1

Tasks	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
Host VM	s_3	s_4	s_3	s_3	s_4	s_4
Frequency	$f_{3,1}$	$f_{4,1}$	$f_{3,1}$	$f_{3,1}$	$f_{4,1}$	$f_{4,1}$
WCET ($\Phi_{i,u}(f)$)	13.81	13.81	13.81	7.34	15.01	11.88
Start Time	0	15.95	13.81	32.02	30.36	45.37
Finish Time	13.81	29.76	27.62	39.36	45.37	57.25

(b) Resultant schedule of the whole Algorithm 1

Tasks	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
Host VM	s_3	s_4	s_3	s_3	s_4	s_4
Frequency	$f_{3,2}$	$f_{4,2}$	$f_{3,2}$	$f_{3,2}$	$f_{4,2}$	$f_{4,2}$
WCET ($\Phi_{i,u}(f)$)	8.75	8.75	8.75	4.96	9.45	7.56
Start Time	0	10.9	8.75	21.91	20.24	29.69
Finish Time	8.75	19.65	17.5	26.87	29.69	37.25

Similar to the inputs of Algorithm 1, the task remapping algorithm has two more parameters (*A* and *FL*) which indicate the initial schedule generated by Algorithm 1. In Algorithm 2, step 1 calculates the energy consumption $E_{\min}$ of the initial schedule as the baseline energy consumption for the following comparison. Step 2 sorts all the tasks by their ranks (see definitions in Eq. (21) in descending order, and saves the result in a priority queue *TPQ*. While *TPQ* is not empty, step 3 dequeues a task with the highest rank and records the index of its current VM host in *temp*. Steps 4–12 tentatively switch tasks among VMs to achieve better energy consumption. Step 4 declares and initializes a variable *dest* to indicate the target VM of τ_u. If *dest* equals -1, it means that there is no need to change the VM for τ_u. Steps 3–10 tries to evaluate the energy consumption of the allocation for τ_u on different VMs. Based on the frequency level information derived by Algorithm 1 and the new task-to-VM mapping indicated by step 5, a new schedule can be inferred. Steps 6–8 calculate the reliability, total energy consumption and makespan of this schedule, respectively. If the new schedule satisfies all the given constraints and has a better energy consumption, steps 9 updates the best energy consumption achieved so far and step 10 assigns τ_u to a better VM (i.e., s_i) that costs less energy. After a try of all VM allocations for τ_u, if $dest \neq -1$, we will update the VM allocation for τ_u permanently. Otherwise, we will keep the VM allocation for τ_u without any change. Finally, step 13 reports the newly updated task-to-VM mapping with better overall energy consumption.

Due to the frequency adjustment in the second part (steps 9–16 of Algorithm 1) of our initial scheduling algorithm, we sacrifice the energy consumption to meet both the completion time and reliability constraints. To reduce the overall energy consumption, the second phase of our approach tries to move tasks to VMs with better energy-efficiency. Fig. 4 shows the remapped schedule generated by Algorithm 2 based on the result in Table 2(b). Since VM s_1 and s_2 have better energy-efficiency, by using Algorithm 2 task τ_4 is move to s_1. The completion time and reliability of the newly obtained schedule are 43.28 s and 0.9991, respectively. The energy consumption of this schedule is 2158 J, which is 5% less than the schedule shown in Table 2(b) without violating the

Input: i) A set of VMs $S = \{s_1, s_2, \ldots, s_n\}$;
ii) A workflow W with task set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_m\}$;
iii) A given target completion time D;
iv) A given target reliability R_{goal};
v) The remapped task-to-VM mapping A;
vi) Frequency levels of all the task-VM pairs, FL;
Output: A remapped task-to-VM mapping A
remap$(S, W, D, R_{goal}, A, FL)$
begin
1. $E_{\min} = E(A, FL^A)$; // Equation (12)
2. $TPQ = SortByRank(W, A, FL)$;
while $!TPQ.isEmpty()$ **do**
3. $\tau_u = TPQ.dequeue()$; $temp = A_u$;
4. $dest = -1$;//destination VM for task τ_u
for $(i = 1; i \leq n \bigwedge i \neq A_u; i{+}{+})$ **do**
5. $A_u = i$;
6. $R = R(A, FL^A)$; // Equation (11)
7. $E_{total} = E(A, FL^A)$; // Equation (12)
8. $M = M(A, FL^A)$; // Equation (14)
if $R_{goal} \leq R \bigwedge M \leq D \bigwedge E_{total} < E_{\min}$ **then**
9. $E_{\min} = E_{total}$;
10. $dest = i$;
end
end
if $dest \neq -1$ **then**
11. $A_u = dest$;
else
12. $A_u = temp$;
end
end
13. return A;
end

Algorithm 2. Task remapping algorithm.

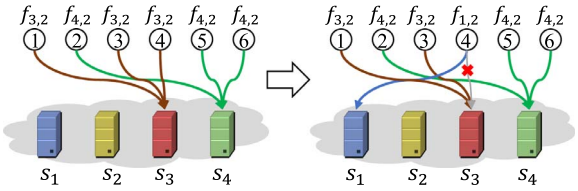


Fig. 4. Resultant schedule of the task remapping algorithm.

given constraints.

4.3. Reliability-aware DVFS algorithm

When generating an initial schedule, our approach focuses on quickly finding a feasible schedule in an iterative manner. In each iteration, Algorithm 1 coarsely increases the frequency of each task-VM pair by one level. Due to the lack of checking every possible combinations of task-VM pairs and frequencies, Algorithm 1 may generate an initial schedule with high energy consumption. Although the second phase of our approach can improve the energy consumption by task remapping, the updated task-to-VM mapping may be still not energy efficient, since the effect of selected task frequencies is not studied in Algorithm 2. Due to the precedence constraints among tasks and the disturbance caused by task remapping process, the schedule generated by Algorithm 2 may have large spare time slots between neighboring tasks on non-critical workflow paths. Such slots can be utilized to enable DVFS, which can save the energy of tasks by scaling down their operating frequencies. Based on these observations, we develop a reliability-aware DVFS method to further optimize energy consumption of the schedule that is generated from the first two phases.

Algorithm 3 presents the details of our reliability-aware DVFS algorithm. Note that for the inputs the task-to-VM mapping is generated by Algorithm 2 and the frequencies of task-VM pairs comes from Algorithm 1. In step 1, we create a task queue Q_i for each VM s_i and sort the tasks based on their temporal relations. Step 2 constructs a priority queue *TPQ* to save all the tasks, which are sorted by their ranks in descending order. For each task τ_u dequeued from *TPQ* in step 3, steps 4–9 calculate its available execution time and steps 10–16 apply DVFS to reduce its energy consumption. To calculate the available execution time for τ_u, we need to figure out two kinds of tasks: the next task that follows τ_u on the same VM, and the immediate successors of τ_u. In step 4, we use lim_1 to denote the start time of task τ_v that follows τ_u on the same VM, use lim_2 to denote the latest possible finish time for τ_u, and use i to indicate the index of the assigned VM for τ_u. In step 5, we calculate the earliest start time of the following task on the same VM and save it in lim_1. If τ_v does not exist, lim_1 will be equal to the given completion time constraint D. In our problem model, each task is assumed to transfer its output data to all of its successors on time. We need to make sure that all the successors of τ_u can start execution no later than their predefined start time. Therefore, to calculate the latest possible finish time of τ_u, we need to investigate the start time of all its successive tasks as shown in steps 6–8. In step 6, we use j to denote the index of τ_v. If τ_u and τ_v are allocated to different VMs, the latest possible finish time of τ_u can be calculated based on the formula presented in step 7 considering the data transfer overhead. Otherwise, if τ_u and τ_v share the same VM, step 8 updates the latest possible finish time for τ_u without considering the communication overhead between τ_u and τ_v. Therefore, the latest possible finish time of τ_u can be calculated as *MIN* (lim_1, lim_2). In step 9, *atime* saves the length of the maximum available time slot for τ_u. After figuring out the maximum execution time slot for τ_u, our approach tries to use DVFS to optimize the energy consumption of τ_u. Step 10 calculates the energy consumption of τ_u without changing its original frequency, which can be used as a baseline for the comparison. Steps 11–16 try each frequency level of VM s_i for task τ_u in an ascending order. Step 11 records the best frequency level information for τ_u on s_i searched so far, and tries to assign a new frequency to the pair of τ_u and s_i. Step 12 calculates the new worst execution time for τ_u based on the frequency update, and step 13 calculate the new energy consumption accordingly. If the new frequency can lead to a better energy consumption, step 15 will update the best energy consumption for τ_u. Otherwise, both the energy and frequency information will not be updated for τ_u as shown in step 16. When all the tasks in *TPQ* have been optimized, step 17 will return a new frequency assignment for the task-to-VM mapping A. Note that Algorithm 3 does not change the VM assignment for any tasks.

In addition to task remapping that reduces energy consumption by finding VMs with better energy-efficiency, DVFS can be used to further lower the overall energy consumption. To achieve this goal, Algorithm 3 tries to scale down the task frequencies by fully utilizing the slack time between tasks. Table 3 shows the updates of task frequencies for the resultant schedule in Fig. 4 after using our DVFS algorithm. In this example, only the frequency level of τ_6 is scaled down. The completion time and reliability of the updated schedule are 47.6 s and 0.998, respectively. The energy consumption of this schedule is 2085 J, which is 3.4% less than the one of the schedule obtained in Fig. 4.

4.4. Schedule generation

Algorithm 4describes the overall process of our approach, which has been illustrated in Fig. 2. In this algorithm, step 1 uses the function *genInitSchedule* implemented in Algorithm 1 to search for an initial task schedule that satisfies both completion time and reliability constraints. Based on the initial schedule, step 2 invokes the function *remap* implemented in Algorithm 2, which can be used to re-allocate tasks to VMs with less energy consumption. To further reduce the overall energy consumption, based on DVFS technique step 3 tries to fully utilize the spare time slots for workflow tasks to scale down their frequencies. Finally, step 4 reports two data structures: the task-to-VM mapping A generated by step 2, and the optimized frequency levels for A (i.e., FL^A) derived by step 3. These two outputs can be used to derive a feasible energy-efficient schedule in the form of $\left\{\left(\tau_u, s_{A_u}, f_{A_u,FL^A_{A_u,u}}\right) | 1 \le u \le m\right\}$.

4.5. Complexity analysis

To analyze the computation complexity of our workflow scheduling approach, we need to take the three phases (i.e., initial scheduling, task remapping, DVFS) into account. Assume that n is the number of available VMs provided by a cloud data center, m is the number of tasks in a given workflow, and k is the maximum number of frequency levels of all the VMs. The computation complexity of the initial scheduling phase is $O(n \times m \times k)$. Note that the computation complexity of HEFT adopted in the first phase is $O(n \times m)$. Since the computation complexity of calculating Eqs. (11) and (12) is $O(m)$, the whole task remapping phase has a computation complexity of $O(n \times m^2)$. For the third phase, the reliability-aware DVFS algorithm has a computation complexity of $O(m^2 \times k)$. Since modern processors have limited number of frequency levels (e.g., ≤ 10), we can treat k as a constant. Therefore, the overall computation complexity of our workflow scheduling is $O(n \times m^2)$.

When the workflow scheduling problem contains a large number of tasks and VMs, it may take a longer time to generate a task schedule using our approach. To reduce this time, proper task clustering method can be adopted. By partitioning workflow tasks into several clusters and scheduling the tasks of each cluster to a set of selected VMs using our approach, we can quickly figure out a schedule for the workflow with reduced energy consumption. Assuming that workflow tasks are partitioned into c clusters, the complexity to achieve an optimized schedule based on task and VM clustering is $O(\frac{n \times m^2}{c^2})$, which needs much less

Input: i) A set of VMs $S = \{s_1, s_2, \ldots, s_n\}$
ii) A workflow W with task set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_m\}$;
iii) A given target completion time D;
iv) A given target reliability R_{goal};
v) The remapped task-to-VM mapping A;
vi) Frequency levels of all the task-VM pairs, FL;
Output: New execution frequencies for all tasks
DVFS$(S, W, D, R_{goal}, A, FL)$
begin
  foreach $s_i \in S$ **do**
    1. $Q_i = TaskQueue(s_i, A)$;
  end
  2. $TPQ = SortByRank(W, A, FL)$;
  while $!TPQ.isEmpty()$ **do**
    3. $\tau_u = TPQ.dequeue()$;
    4. $lim_1 = lim_2 = D$; $i = A_u$;
    if $\exists\ \tau_v$. τ_v *next to task* τ_u *in* Q_i **then**
      5. $lim_1 = ST_{i,v}(f_{i,FL_{i,v}})$;
    end
    foreach $\tau_v \in succ(\tau_u)$ **do**
      6. $j = A_v$;
      if $i \neq j$ **then**
        7. $lim_2 = MIN(lim_2, ST_{j,v}(f_{j,FL_{j,v}}) - \frac{w_{u,v}}{b_{i,j}})$;
      else
        8. $lim_2 = MIN(lim_2, ST_{j,v}(f_{j,FL_{j,v}}))$;
      end
    end
    9. $atime = MIN(lim_1, lim_2) - ST_{i,u}(f_{i,FL_{i,u}})$;
    10. $E_{\min} = E_{i,u}(FL_{i,u})$; // Equation (10)
    foreach $f_{i,l} \in F_i$ *in ascending order* **do**
      11. $temp = FL_{i,u}$; $FL_{i,u} = l$;
      12. $\Phi_{i,u} = \Phi_{i,u}(f_{i,l})$; // Equation (3)
      13. $E_{i,u} = E_{i,u}(f_{i,l})$; // Equation (10)
      14. $R = R(A, FL)$; // Equation (11)
      if $R_{goal} \leq R \bigwedge \Phi_{i,u} \leq atime \bigwedge E_{i,u} < E_{\min}$ **then**
        15. $E_{\min} = E_{i,u}$;
      else
        16. $FL_{i,u} = temp$;
      end
    end
  end
  17. return FL;
end

Algorithm 3. Reliability-aware DVFS algorithm.

Table 3
Frequency updates by the DVFS algorithm.

Tasks	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
Pre-DVFS Frequencies	$f_{3,2}$	$f_{4,2}$	$f_{3,2}$	$f_{1,2}$	$f_{4,2}$	$f_{4,2}$
Post-DVFS Frequencies	$f_{3,2}$	$f_{4,2}$	$f_{3,2}$	$f_{1,2}$	$f_{4,2}$	$f_{4,1}$

Table 4
Voltage/Frequency/Power Settings of AMD Processors.

AMD Turion MT-34			AMD Opteron 2218		
Frequency	Voltage	Power	Frequency	Voltage	Power
0.8 GHz	0.90 V	6.25 W	1.0 GHz	1.10 V	26.16 W
1.0 GHz	1.00 V	9.65 W	1.8 GHz	1.15 V	51.47 W
1.2 GHz	1.05 V	12.76 W	2.0 GHz	1.15 V	57.19 W
1.4 GHz	1.10 V	16.34 W	2.2 GHz	1.20 V	68.49 W
1.6 GHz	1.15 V	20.41 W	2.4 GHz	1.25 V	81.08 W
1.8 GHz	1.20 V	25.00 W	2.6 GHz	1.30 V	95.00 W

schedule generation time than the scheduling method without task clustering.

5. Performance evaluation

To evaluate the effectiveness of our approach, we investigated three types of well-known workflows [32] from different scientific areas: (i) *CyberShake* from the earthquake science filed, (ii) *LIGO Inspiral* from the astrophysics area, and (iii) *Montage* from the astronomy field. For each type, we consider two workflows with different scales, i.e., small workflows that consist of 30 tasks and large workflows that have 60 tasks for each workflow. All these workflows were generated by the toolkit *WorkflowGenerator* [33] based on its default configurations.

Based on the generated workflows, this section compares the performance of our approach with two promising DVFS techniques, i.e., DEWTS [21] and EES [20] from the perspective of energy consumption with different completion time and reliability requirements. The DEWTS algorithm has three stages. Firstly, DEWTS tries to efficiently allocate all the given tasks to a set of VMs using the HFET algorithm without considering DVFS. Secondly, DEWTS greedily consolidates tasks to as few VMs as possible. Finally, by fully utilizing the slack time based on task scaling and reassigning, DEWTS can achieve a task-to-VM allocation solution with optimized power consumption. Similar to DEWTS, the EES algorithm has two phases. In the first phase, EES prioritizes the tasks and generates a task-to-processor mapping using the HEFT algorithm without considering DVFS. In the second phase, EES improves the energy consumption of the mapping based on DVFS by scheduling the nearby tasks of a non-critical jobs with a uniform frequency. Note that the original DEWTS and EES methods focus on minimizing the overall energy consumption while guaranteeing the completion time of workflows. None of them support the workflow reliability analysis. To enable fair comparison, we modified these two approaches to incorporate the reliability modeling and evaluation considering the effects of DVFS. All the experiments were performed on a desktop with 3.10GHz Intel Core i5 CPU and 10GB RAM.

5.1. Experimental setup

We adopted the cloud simulator CloudSim [34] to simulate a cloud data center which consists of 20 VMs running on two different types of physical processors. In the experiments, we assumed that each VM executes on one individual physical processor. We let half of the VMs execute on AMD Turion MT-34 processors and the other half execute on AMD Opteron 2218 processors. Both kinds of AMD processors support DVFS operations during VM executions. Table 4 presents the voltage

Input: i) A set of VMs $S = \{s_1, s_2, \ldots, s_n\}$;
ii) A workflow W with task set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_m\}$;
iii) A given target completion time D;
iv) A given target reliability R_{goal};
Output: A feasible energy-efficient task schedule
genSchedule(S, W, D, R_{goal})
begin
1. (A, FL)=genInitSchedule(S, W, D, R_{goal});
2. A=remap(S, W, D, R_{goal}, A, FL);
3. FL=DVFS(S, W, D, R_{goal}, A, FL);
4. return (A, FL^A);
end

Algorithm 4. Overall process of our approach.

Table 5
Reliability of workflow schedules generated by HEFT.

	CyberShake	LIGO Inspiral	Montage
Small	0.9997	0.9998	0.9999
Large	0.9995	0.9995	0.9997

and corresponding frequency settings [35,36] of these two kinds of processors. Based on these settings, the power consumption of a VM running at different frequencies can be approximated using Eq. (8) which is adopted in [37]. Note that an AMD Turion MT-34 only has one core, while an AMD Opteron 2218 processor has two cores. Therefore, the full processing capacity of an AMD Turion MT-34 processor (with 1 hardware thread) is 1800 MIPS, and the full processing capacity of an AMD Opteron 2218 processor (with 2 hardware threads) is 5200 MIPS. In the experiments, we consider the communication delays among VMs. We set the maximum network bandwidth between two VMs to 1 Gbps.

Similar to the experimental settings presented in [5,38], in our experiments the checkpoint information is saved in secure devices. We set the time overhead of each checkpoint (i.e., O_{cp}) to 0.5 s. We assume that for both types of physical processors the occurrence of transient faults follows the same Poisson distribution, where the constant parameters $\lambda_{i,0}$ and ξ_i ($1 \le i \le n$) defined in Eq. (5) are set to 10^{-6} and 2, respectively. For the worst case, we set the maximum number of transient faults K_u ($1 \le u \le m$) during the execution of task τ_u to 2.

To achieve the tightest possible initial schedules, in the first phase of our approach, we adopt the HEFT method which does not take DVFS into account. Since HEFT assumes that all the VMs are running at the maximum processor frequency levels, it can generate schedules with high reliability. For example, Table 5 shows the reliability information of the schedules generated by HEFT considering the effects of checkpointing and rollback-recovery techniques. From this table, we can find that HEFT can achieve high reliability ($\ge$ 0.9995) for all the workflows with different types and scales.

Although the HEFT approach can simultaneously maximize the reliability and minimize the makespan of a workflow, the task execution at the highest frequencies will inevitably result in high energy consumption. To make a tradeoff between energy consumption and reliability, our approach tries to exploit the potential of DVFS to save workflow energy consumption under given makespan constraints. In the experiments, we use HEFT as the baseline approach for the comparison of power consumption. We set the desired workflow completion time to $D = \beta M_H$, where M_H denotes the makespan of a workflow schedule derived by HEFT and β ($\ge$ 1) is a constant real number. To check the performance of our approach under different completion time constraints, in the experiments we set β to 1.2, 1.4, 1.6 and 1.8, respectively.

5.2. Results and analysis

To validate the effectiveness of our approach, we compare the energy consumption of workflows under the constraints of reliability and completion time, respectively.

5.2.1. Results of workflows with fixed reliability goals

Figs. 5 and 6 compare the overall energy consumption of schedules generated from the three workflows (i.e., *CyberShake, LIGO Inspiral*, and *Montage*) under the constraints of a fixed reliability but different completion time. In these two experiments, we set the reliability constraint R_{goal} to 0.99. We used the makespan of schedules achieved by HEFT as the baseline of completion time requirements, and used β to indicate the relative length of completion time requirements. For example, $\beta = 1.2$ indicates that the makespan of target schedules cannot be 1.2 times longer than the makespan of schedules derived by the HEFT approach. Note that since HEFT approach focuses on finding minimum makespan schedules without considering DVFS, for a specific workflow the schedules derived using the HEFT approach with different completion time constraints are the same.

In Fig. 5, we only generated one small workflow for each sub-figure. Since all the three small workflows have the same number of tasks, their schedule generation time for different workflow task scheduling

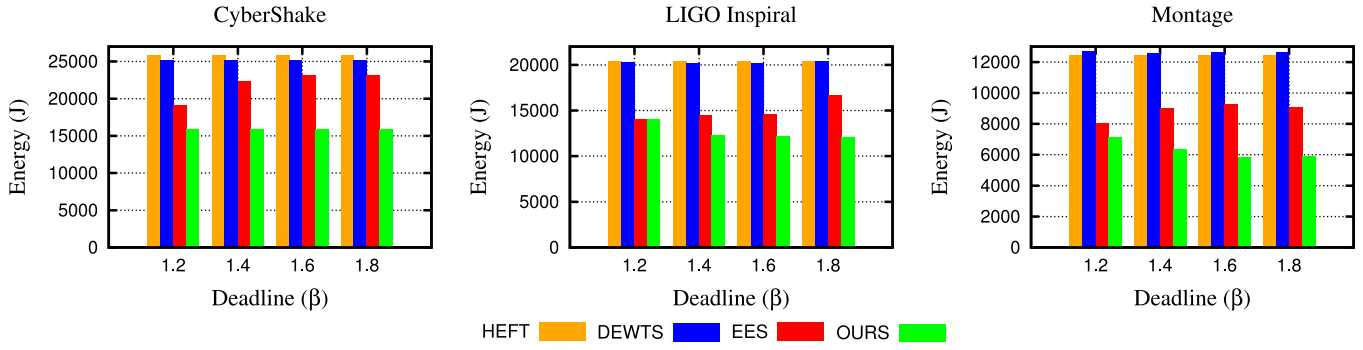


Fig. 5. Energy consumption of small workflows under the constraints of a fixed reliability (R_{goal} = 0.99) and varying completion time.

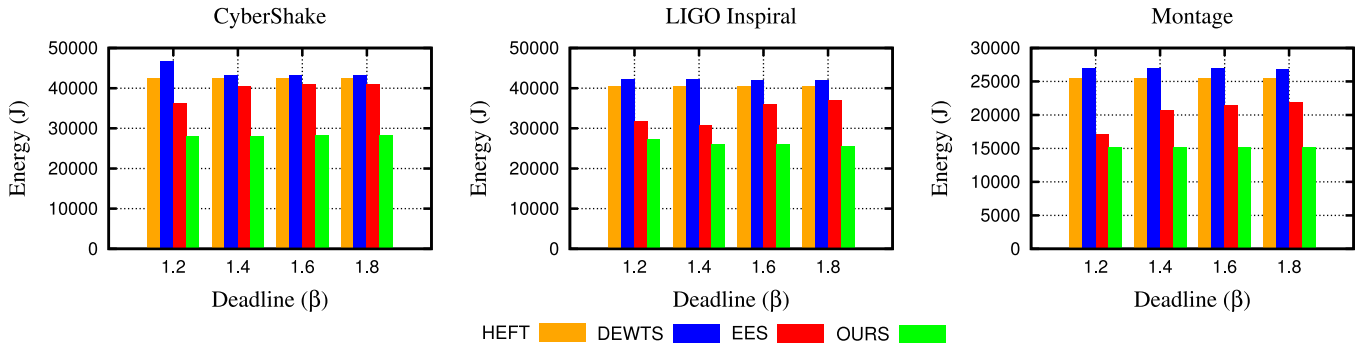


Fig. 6. Energy consumption of large workflows under the constraints of a fixed reliability (R_{goal} = 0.99) and varying completion time.

methods is quite similar. By using our approach, it costs around 1.1 s on average to generate one schedule for a given workflow. Similar to Fig. 5, Fig. 6 investigates the performance of our approach on three large workflows, each of which consists of 60 tasks. By using our proposed approach, it costs around 3.2 s to obtain a feasible schedule on average. Note that the DEWTS approach is not good at dealing with large workflows. When handling small workflows, the performance of DEWTS is similar to the one of HEFT. However, when handling large workflows, its performance is worse than the HEFT approach though DVFS is enabled. For example, when dealing the large *CyberShake* workflow with $\beta = 1.2$, the power consumption of the schedule derived by DEWTS is 10% worse than the one generated by HEFT.

From Figs. 5 and 6, we can find that our approach significantly outperforms the methods HEFT, DEWTS and EES. As an example of *CyberShake* workflow shown in Fig. 5, although EES method outperforms both HEFT and DEWTS approaches in all the cases in terms of power consumption, its performance is still far behind our approach. When we adopt the completion time constraint with $\beta = 1.8$, our proposed approach can save 31.3% energy as compared with EES method. With the same constraints, our approach can save 37.1% energy as compared with DEWTS method. From Fig. 5, we can find that DEWTS does not improve the energy consumption too much as compared with HEFT. In Fig. 6, the energy consumption of schedules generated by DEWTS is worse than the one of schedules generated by HEFT. This is because DEWTS tries to consolidate more tasks on a VM to reduce the amount of VMs used for workflow execution. To complete tasks in time, DEWTS has to increase execution frequencies of tasks, which leads to high energy consumption.

When we relax completion time constraints in both Figs. 5 and 6 with larger β, we can observe that the schedules generated by our approach require less energy in a consistent manner. However, this trend does not hold for the EES approach. We can even observe an uptrend of energy consumption for schedules with looser completion time constraints in both Figs. 5 and 6. In other words, when completion time is not a major concern in workflow scheduling with a fixed reliability constraint, EES approach cannot guarantee its effectiveness in finding schedules with small energy consumption. The reason of this phenomenon is primarily because EES tends to assign the lowest possible operating frequencies to workflow tasks. When the constraint of workflow completion time extends, more tasks suffer from low reliability due to their low operating frequencies. To guarantee the specified reliability by using checkpointing and rollback-recovery techniques, the overall workflow execution time will be prolonged. In this case, the increased idle time between tasks together with lowered CPU utilization may lead to more waste on static power. In contrast, our approach considers the impacts of frequency scaling on task execution time and scales down operating frequency for a task only if its energy consumption can be decreased. Therefore, when the workflow deadline constraint extends, the energy consumption of schedules generated by our approach will not increase. Moreover, since our approach allows task remapping among physical processors in the second phase, the overall energy consumption of workflows can be further optimized.

5.2.2. Results of workflows with fixed deadlines

Figs. 7 and 8 compare the overall energy consumption of schedules generated from the three workflows with a fixed completion time constraint but different reliability requirements. For each workflow in both figures, we set the completion time constraint using $\beta = 1.4$, and we investigated the energy consumption of schedules generated by different approaches with four reliability requirements, i.e., $R_{goal} = 0.990$, $R_{goal} = 0.992$, $R_{goal} = 0.994$ and $R_{goal} = 0.996$. Note that in the same sub-figure, the schedules generate by HEFT have a reliability larger than 0.999 (see details in Table 5). Fig. 7 presents the results for small workflows. Since the workflow size is the same and reliability requirements are quite similar, the schedule generation time for different scheduling approaches here is almost the same. To achieve a feasible schedule for a small workflow under both completion time and reliability constraints, our approach costs around 0.92 s. Similarly, to achieve a feasible schedule for a large workflow as shown in Fig. 7, our approach costs around 3.14 s on average.

From these figures, we can find that the schedules generated by our approach consume much less energy than their counterparts generated

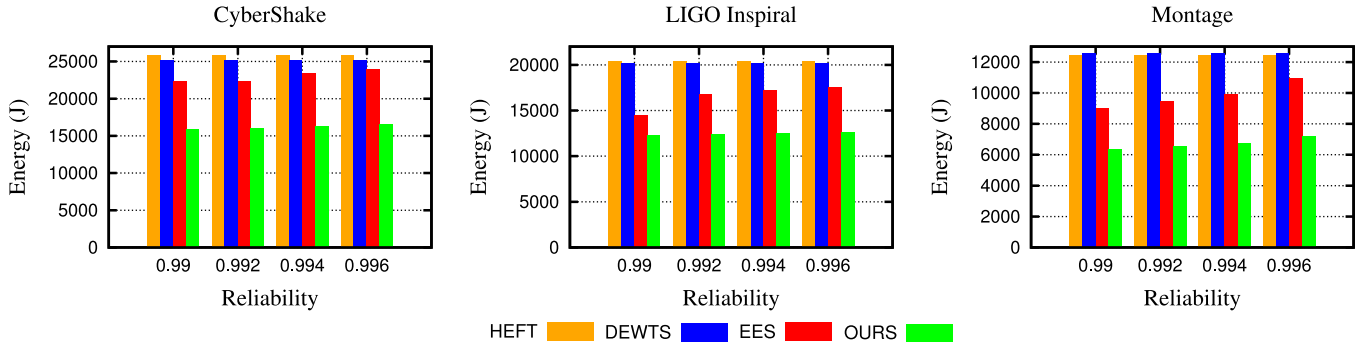


Fig. 7. Energy consumption of small workflows under the constraints of a fixed completion time ($\beta = 1.4$) and varying reliability.

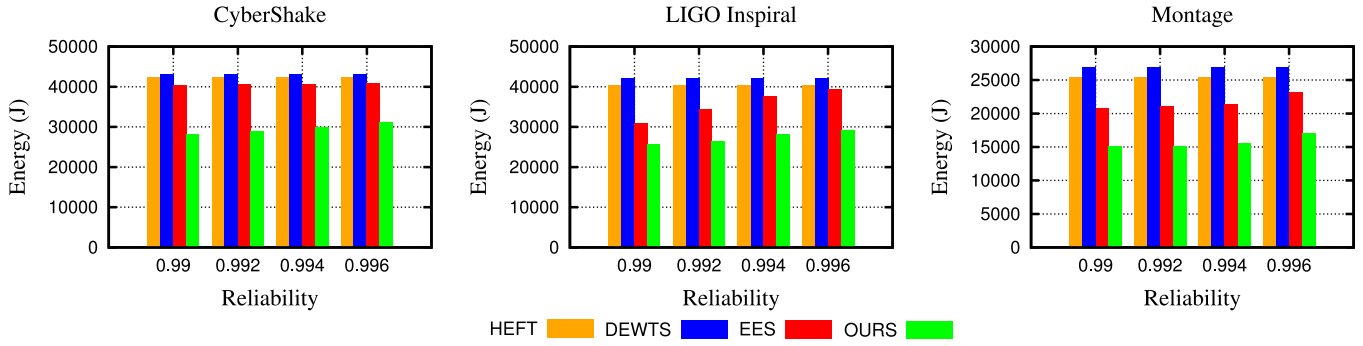


Fig. 8. Energy consumption of large workflows under the constraints of a fixed completion time ($\beta = 1.4$) and varying reliability.

by HEFT, DEWTS and EES. As an example of the *Montage* workflow shown in Fig. 7, when the reliability requirement is 0.996, our approach can reduce energy consumption by 34.1% as compared to EES. Moreover, as the reliability target goes high, from these two figures we can find a trend of increasing the overall energy consumption for each workflow, though the difference is negligible for the DEWTS method. This is mainly because when the completion time constraint is fixed but the reliability target goes high, we need to increase the frequency levels for tasks in order to reduce the average fault arrival rate during their executions.

6. Conclusion

As a popular power management technique, DVFS has been increasingly adopted in cloud data centers to reduce the energy consumption of workflow applications. However, due to the scaling of technology, improper selection of CPU frequency for VMs may drastically increase the susceptibility of cloud data centers to soft errors, which may significantly impact the QoS and reliability of workflow executions. Therefore, how to accomplish workflow applications on time with less energy in the presence of soft errors is becoming a key issue for cloud service providers. To address this problem, we proposed a novel three-phase approach that can generate energy-efficient task schedules for cloud workflows under both reliability and completion time constraints. Comprehensive experimental results obtained from various scientific workflow benchmarks show the effectiveness of our approach. Compared to state-of-the-art approaches, our approach can save more than 30% energy without violating reliability and completion time constraints.

In this paper, each task in a workflow is assigned with a fixed execution time. However, this is not always true in many scenarios. For example, due to performance variations, the execution time of the same task on different processors of the same type can be different [39,40]. In the future, we plan to take such factors into account to make our approach more adaptive to practical complex scenarios. Meanwhile, besides power consumption, the operating cost is another major concern of cloud service providers. Within a DVFS-enabled cloud data center, how to optimize the cost of a given workflow while satisfying specific deadline and reliability requirements is also an interesting topic that is worthy of further study.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.sysarc.2018.03.001.

References

[1] X. Liu, D. Yuan, G. Zhang, W. Li, D. Cao, Q. He, J. Chen, Y. Yang, The Design of Cloud Workflow Systems, Springer, 2012.

[2] U. Wajid, C. Cappiello, P. Plebani, P. Sampaio, On achieving energy efficiency and reducing CO_2 footprint in cloud computing, IEEE Trans. Cloud Comput. 4 (2) (2016) 138–151.

[3] L. Wang, G. Laszewski, J. Dayal, F. Wang, Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS, Proceedings of International Conference on Cluster, Cloud and Grid Computing (CCGrid), (2012), pp. 368–377.

[4] X. Lin, Y. Wang, Q. Xie, M. Pedram, Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment, IEEE Trans. Serv. Comput. 8 (2) (2015) 175–186.

[5] D. Zhu, R. Melhem, D. Mosse, The effects of energy management on reliability in real-time embedded systems, Proceedings of International Conference on Computer-Aided Design (ICCAD), (2004), pp. 35–40.

[6] S. Mukherjee, Architecture Design for Soft Errors, Elsevier, 2008.

[7] K.V. Vishwanath, N. Nagappan, Characterizing cloud computing hardware reliability, Proceedings of ACM Symposium on Cloud Computing (SoCC), (2010), pp. 193–204.

[8] Z. Li, S. Ren, G. Quan, Energy minimization for reliability-guaranteed real-time applications using DVFS and checkpointing techniques, J. Syst. Archit. 61 (2) (2015) 71–81.

[9] H. Topcuouglu, S. Hariri, M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.

[10] F. Cao, M. Zhu, Energy efficient workflow job scheduling for green cloud, Proceedings of International Symp. on Parallel and Distributed Processing Workshops and PhD Forum, (2013), pp. 2218–2221.

[11] J.J. Durillo, V. Nae, R. Prodan, Multi-objective energy-efficient workflow scheduling using list-based heuristics, Future Gen. Comput. Syst. 36 (2014) 221–236.

[12] D. Zhu, H. Aydin, Energy management for real-time embedded systems with reliability requirements, Proceedings of International Conference on Computer-Aided Design (ICCAD), (2006), pp. 528–534.

[13] X. Wang, R. Buyya, J. Su, Reliability-oriented genetic algorithm for workflow applications using max-min strategy, Proceedings of International Symposium on Cluster Computing and the Grid (CCGrid), (2009), pp. 108–115.

[14] C. Yan, Z. Hu, X. Li, Z. Hu, P. Xiao, Reliability enhanced grid workflow scheduling algorithm with budget constraint, Proceedings of International Conference on Information and Automation, (2011), pp. 629–633.

[15] L. Zhang, K. Li, C. Li, K. Li, Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems, J. Inf. Sci. 379 (2017) 241–256.

[16] M. Sedaghat, E. Wadbro, J. Wilkes, S.D. Luna, O. Seleznjev, E. Elmroth, Diehard: reliable scheduling to survive correlated failures in cloud data centers, Proceedings of International Symposium on Cluster, Cloud and Grid Computing (CCGrid), (2016), pp. 52–59.

[17] Y. Gao, S.K. Gupta, Y. Wang, M. Pedram, An energy-aware fault tolerant scheduling framework for soft error resilient cloud computing systems, Proceedings of Design, Automation and Test in Europe (DATE), (2014), pp. 24–28.

[18] A. Zhou, S. Wang, Z. Zheng, C.H. Hsu, M.R. Lyu, F. Yang, On cloud service reliability enhancement with optimal resource usage, IEEE Trans. Cloud Comput. 4 (4) (2016) 452–466.

[19] J. Jiang, Y. Lin, G. Xie, L. Fu, J. Yang, Time and energy optimization algorithms for the static scheduling of multiple workflows in heterogeneous computing system, J. Grid Comput. Appear (2017).

[20] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, X. Huang, Enhanced energy-efficient scheduling for parallel applications in cloud, Proceedings of International Symposium on Cluster, Cloud and Grid Computing (CCGrid), (2012), pp. 781–786.

[21] Z. Tang, L. Qi, Z. Cheng, K. Li, S.U. Khan, K. Li, An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment, J. Grid Comput. 14 (1) (2016) 55–74.

[22] M. Salehi, M. Tavana, S. Rehman, M. Shafique, A. Ejlali, J. Henkel, Two-state checkpointing for energy-efficient fault tolerance in hard real-time systems, IEEE Trans. Very Large Scale Integr. Syst. 24 (7) (2016) 2426–2437.

[23] M.A. Rodriguez, R. Buyya, Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds, IEEE Trans. Cloud Comput. 2 (2) (2014) 222–235.

[24] T. Wei, X. Chen, S. Hu, Reliability-driven energy-efficient task scheduling for multiprocessor real-time systems, IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 30 (10) (2011) 1569–1573.

[25] Y. Zhang, K. Chakrabarty, A unified approach for fault tolerance and dynamic power management in fixed-priority real-time embedded systems, IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 25 (1) (2006) 111–125.

[26] J. Zhou, T. Wei, Stochastic thermal-aware real-time task scheduling with considerations of soft errors, J. Syst. Softw. 102 (2015) 123–133.

[27] R. Ge, X. Feng, K.W. Cameron, Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters, Proceedings of International Conference on Supercomputing, (2005). 34–34.

[28] T. Guerout, T. Monteil, G.D. Costa, R.N. Calheiros, R. Buyya, M. Alexandru, Energy-aware simulation with DVFS, Simul. Modell. Pract. Theory 39 (2013) 76–91.

[29] C. Piguet, C. Schuster, J.L. Nagel, Optimizing architecture activity and logic depth for static and dynamic power reduction, Proceedings of Northeast Workshop on Circuits and Systems, (2004), pp. 41–44.

[30] B. Zhao, H. Aydin, D. Zhu, Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints, ACM Trans. Des. Autom. Electron. Syst. 18 (2) (2013).

[31] D. Meisner, B. Gold, T. Wenisch, Powernap: eliminating server idle power, Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), (2009), pp. 205–216.

[32] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.H. Su, K. Vahi, Characterization of scientific workflows, Proceedings of Workshop on Workflows in Support of Large-Scale Science, (2008), pp. 1–10.

[33] R.F.d. Silva, W. Chen, G. Juve, K. Vahi, E. Deelman, Community resources for enabling research in distributed scientific workflows, Proceedings of International Conference on e-Science, (2014), pp. 177–184.

[34] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F.D. Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, J. Softw. 41 (4) (2011) 23–50.

[35] L. Wang, G. Laszewski, J. Dayal, F. Wang, Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS, Proceedings of International Conference on Cluster, Cloud and Grid Computing (CCGrid), (2010), pp. 368–377.

[36] R. Ge, X. Feng, W.C. Feng, K.W. Cameron, CPU miser: a performance-directed, run-time system for power-aware clusters, Proceedings of International Conference on Parallel Processing (ICPP), (2007), pp. 18–26.

[37] W. Liu, W. Du, J. Chen, W. Wang, G. Zeng, Adaptive energy-efficient scheduling algorithm for parallel tasks on homogeneous clusters, J. Netw. Comput. Appl. 41 (2014) 101–113.

[38] S. Di, Y. Robert, F. Vivien, D. Kondo, C.L. Wang, F. Cappello, Optimization of cloud

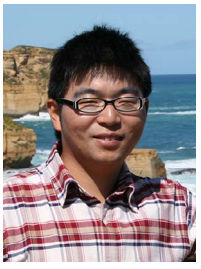
task processing with checkpoint-restart mechanism, Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis, (2013), pp. 1–12.
[39] M. Chen, X. Fu, S. Huang, X. Liu, J. He, Statistical model checking-based evaluation and optimization for cloud workflow resource allocation, IEEE Trans. Cloud Comput. Appear (2017).
[40] S. Huang, M. Chen, X. Liu, D. Du, X. Chen, Variation-aware resource allocation evaluation for cloud workflows using statistical model checking, Proceedings of International Conference on Big Data and Cloud Computing (BDCloud), (2014), pp. 201–208.

Tingming Wu received the B.E. degree from the Software Engineering Institute, East China Normal University, Shanghai, China, in 2015. He is currently a master student in the Institute of Computer Science and Software Engineering, East China Normal University. His research interests are in the area of cloud computing, parallel and distributed systems, design automation of embedded systems, and software engineering.

Haifeng Gu received the B.E. degree from the Department of Computer Science and Technology, Sichuan Normal University, Chengdu, China, in 2013. He is currently a Ph.D. student in the Department of Embedded Software and System, East China Normal University, Shanghai, China. His research interests are in the area of Hardware/Software co-validation, symbolic execution, statistical model checking, and software testing.

Junlong Zhou is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, East China Normal University, Shanghai, China. He was a Research Visitor with the University of Notre Dame, Notre Dame, IN, USA. His current research interests include real-time embedded systems, cloud computing, and cyber-physical systems. Mr. Zhou is an Active Reviewer of several international journals, including IEEE Transactions on Computers, IEEE Transactions on Industrial Informatics, Journal of Systems and Software, and Journal of Circuits, Systems, and Computers.

Tongquan Wei received his Ph.D. degree in Electrical Engineering from Michigan Technological University in 2009. He is currently an Associate Professor in the Department of Computer Science and Technology at the East China Normal University. His research interests are in the areas of green and reliable embedded computing, cyber-physical systems, parallel and distributed systems, and cloud computing. He serves as a Regional Editor for Journal of Circuits, Systems, and Computers since 2012. He also served as Guest Editors for several special sections of IEEE TII and ACM TECS.



Xiao Liu received his master degree in management science and engineering from Hefei University of Technology, Hefei, China, 2007, and received his Ph.D. degree in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Australia, 2011. He worked as an associate professor in Software Engineering Institute at East China Normal University from 2013 to 2015. He is currently a senior lecturer at School of Information Technology, Deakin University, Melbourne, Australia. His research interests include software engineering, workflow management systems and cloud computing.

Mingsong Chen received the B.S. and M.E. degrees from Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2003 and 2006 respectively, and the Ph.D. degree in Computer Engineering from the University of Florida, Gainesville, in 2010. He is currently a Professor with the Computer Science and Software Engineering Institute at East China Normal University. His research interests are in the area of design automation of cyber-physical systems, parallel and distributed systems, formal verification techniques and cloud computing. He is an Associate Editor of IET Computers & Digital Techniques, and Journal of Circuits, Systems and Computers.