Chapter 4

A heuristic-based scheduling algorithm for energy and reliability optimization

This chapter[1] proposes a heuristic for workflow scheduling called energy-efficient and reliability-aware workflow task scheduling (EERS) in clouds. The EERS addresses objectives such as energy consumption and system reliability. It consists of five sub-algorithms such as task rank calculation, task clustering algorithm, sub-target time distribution, cluster-VM mapping, and slack algorithm. The EERS provides a trade-off between energy utilization and reliability. The efficiency of the algorithm is evaluated using WorkflowSim tool and well-known scientific workloads like CyberShake and Montage. The experimental results show that it surpassed the related state-of-the-art works in terms of energy-saving and reliability.

4.1 Introduction

Most of the business and complex scientific applications have been using workflows to analyze complex data sets and conduct simulation experiments constructively [7]. A scientific workflow involves individual data transformations and analysis steps, and

[1]This chapter is derived from: Medara, Rambabu, and Ravi Shankar Singh. "Energy Efficient and Reliability Aware Workflow Task Scheduling in Cloud Environment." *Wireless Personal Communications* (2021): 1-20.

mechanisms to link them according to the data dependencies among them [104]. Several scientific applications comprise numerous tasks with precedence constraints and are highly complex with various sizes of tasks. Therefore, scientific workflow management mostly deals with large volumes of data. Such complex workflow applications demand high computing power and high system reliability. With the rapid proliferation of accessing scientific applications, it is necessary to focus on developing the energy-aware workflow scheduling model in a cloud environment.

The scheduling of workflow tasks in distributed platforms such as grids and cloud environments has been extensively considered for many years. Researchers have developed algorithms geared towards different environments: from small-scale homogeneous clusters to large-scale community grids, to the contemporary paradigm, heterogeneous, utility-based, and resource-rich cloud computing [7]. Clouds provide unlimited computing power which is a more relevant platform to execute complex workflow applications. But the computational systems in cloud data centers are not failure-free and any type of fault may be critical to the running application [105]. Particularly, the transient faults increase by the DVFS approach. Such failures may impact the execution nature of the program and provoke unpredictable results [106]. In this work, we proposed an efficient heuristic for energy efficiency and reliability aware workflow tasks scheduling in a cloud environment (EERS) which maximizes the reliability for workflow tasks while conserving energy.

The rest of the chapter is organized as follows. The related work is discussed in Section 4.2. We propose system architecture and models in Section 4.3. Then Section 4.4 presents the five sub-algorithms and EERS algorithm implementation. Section 4.5 gives the algorithm performance evaluation and finally conclusion and future work discussed in Section 4.6.

4.2 Related work

In the past few years, notable attention has been paying to develop the scheduling and energy-efficient resource management models for workflow execution in cloud environments. This problem's efficiency depends on various factors like the arrival rate of the user requests, resource availability, and workloads.

Numerous scheduling algorithms for the cloud environment aimed to optimize the makespan and cost of workflow applications. A fuzzy dominance-based HEFT algorithm to address the cost and makespan of workflow applications in clouds [107], which uses real-world pricing and resource models. A cost-aware large-scale workflow scheduling approach in [108] with DAG splitting mechanism reduces the monetary cost of application by maximizing the VM utilization. A list-based heuristic called Heterogeneous Earliest Finish Time (HEFT) [92] approach gives the best makespan. A heuristic called Jaya algorithm in [109] used to reduce both the computation and communication costs while scheduling the workflow tasks in the cloud environment. A Case Library and Pareto Solution-based hybrid Genetic Algorithm (CLPS-GA) [110] has focused on two objectives such as makespan and energy conservation while scheduling the workflow tasks. It relies on a case library and multiparent crossover operator to effectively ensure the stability, diversity, and convergence in the solution. An approach in [111] has employed an immune genetic algorithm to resolve the QoS constraint satisfaction by considering five objectives for workflow scheduling in the cloud environment. The deadline-based workflow scheduling algorithm [112] employs the Particle Swarm Optimization (PSO) technique to diminish the overall execution cost while scheduling the scientific workflow applications in the Infrastructure-as-a-Service (IaaS) clouds without violating the deadline constraints. However, these works are not considered either energy or reliability objectives.

A green energy-efficient scheduling approach [113] employs the DVFS technique which enables the computing processors to run the task at low voltages and low frequencies. It effectively utilizes the cloud data center resources through task to-VM allocation based on the task dependencies of an application to reduce energy consumption and makespan. An energy-efficient heuristic is proposed in [50] to schedule real-time workflow applications in clouds. It saves energy by effectively utilizing the schedule gaps using per-core DVFS and approximate computations. Similar work in [46] addressed energy, monetary cost, and quality of service objectives. A polynomial-time multi-objective heuristic proposed in [49] to schedule time-constrained tasks on the cloud environment, which optimize energy, cost, and resource utilization while maintaining Service Level Agreements (SLAs). Stackelberg game based Game-Score simulator developed in [114] to schedule the tasks in the cloud environment, which trade-off between energy consumption and schedule length of a workflow. A load balancing-based approach for scheduling tasks in a distributed cloud environment in [44], optimizes resource utilization by estimating task execution time

based on the cloud status and queuing model used to improve response time. A PSO-based multi-objective approach for workflow scheduling in clouds optimizes cost, schedule length, and resource utilization while considering system reliability [115]. From the review of the existing state-of-art scheduling strategies, it is observed that most of these strategies focus on multi-optimization without considering the reliability of the system.

4.3 System models

This section presents the system models such as the cloud datacenter model, application model, power model, and reliability model.

4.3.1 Datacenter Model

The cloud data center is assumed having M heterogeneous physical machines (PM) $PM = \{pm_1, pm_2, \ldots.., pm_M\}$. Every PMs is DVFS enabled and can operate at varying frequency (f) levels $(f_1, f_2, \ldots.., f_k)$, where k is the number of frequency levels. The switching time among these frequency levels approximately takes 10–150μm which is insignificant [47]. The DVFS approach operates processor frequencies under various voltage levels. Every PM is describing with different types of resources such as CPUs, memory capacity, bandwidth, network I/O, and the storage size. These physical machines virtualized into v number of virtual machines (VMs) each of which considered operating at a different frequency level (DVFS enabled) and PM operating frequency attribute to its VMs. A VM can be characterized with maximum computing performance in a Million Instructions Per Second (MIPS), bandwidth (B) etc. Hence, the cloud workflow scheduler has distinct possibilities in selecting the suitable VM to execute a task by meeting workflow constraints. Generally, we consider computing performance relates to the processor frequency. A k^{th} virtual machine vm_k at some level of operating frequency represented as f_{op}^k.

4.3.2 Application Model

In general, a workflow W with dependencies among tasks $Tw = \{t_1, t_2, \ldots .t_n\}$ is modeled as a DAG. A DAG $W = (T_w, C)$ where C is the set of edges or directed arcs represents the dependencies among tasks. An edge c_{ij} is the dependency from t_i to t_j where t_i is one of the parents of t_j and t_j is one of the children of t_i. A task without a parent(s) or predecessor(s) is called an entry task t_{entry} and a task without child(s) or successor(s) is called exit task t_{exit}. A task is prompt to execute when all required resources of its available. When the job t_i completes its execution and its generated output transfers to its children. The data transferred (in MB) from task t_i to its child t_j has represented the weight on the edge c_{ij} as $w(c_{ij})$. We denote the overall execution time of workflow (makespan) as T_M and associated workflow deadline as T_D. For any workflow execution deadline T_D is describe as a time constraint and is user-defined.

The data communication time $T(t_{ij})$ between two precedence constraint tasks is calculated as in equation (4.1)

$$T(t_{ij}) = \frac{w(c_{ij})}{B} \tag{4.1}$$

where B is the bandwidth and $w(c_{ij})$ is the data (in MB) communicated between tasks t_i and t_j. The execution cost of any task t_i calculated as in equation (4.2)

$$T(t_i, vm_k) = \frac{w_i}{f^k_{max}} + T(t_{ij}) \tag{4.2}$$

where $T(t_i, vm_k)$ is the task t_i execution time on vm_k and it includes effective execution time and data communication time. The mean execution time of task t_i is the mean of execution times on various available VMs and is calculated as follows

$$\overline{T}(t_i) = \sum_{k=1}^{n} \frac{T(t_i, vm_k)}{n} \tag{4.3}$$

where n is the number of VMs. The earliest start time EST and as well as earliest finish time EFT of task t_i are calculated as follows

$$EST(t_i) = \begin{cases} 0 & \text{if } t_i = t_{entry} \\ max_{t_p \in parent(t_i)} EFT(t_p) & \text{otherwise} \end{cases} \tag{4.4}$$

$$EFT(t_i) = EST(t_i) + T(t_i, vm_k) \tag{4.5}$$

The $EFT(t_{exit})$ is the minimum makespan of the W $minT_M = EFT(t_{exit})$. Without loss of generality, we consider user defined deadline T_D should be greater than the $minT_M$ i.e. $T_D > minT_M$.

4.3.3 Power Model

The power utilization of computational servers of cloud data centers is due to CPU, memory, network interfaces, storage disks, and other underlying circuits. In comparison with other computing resources, the CPU dominates energy consumption. For this work, we have used the energy model discussed in Section 3.3.4. The energy consumption of a specific task t_i with computation cost w_i, executing on vm_k with operating frequency f_{max} calculated as follows

$$E(f_{op}^{i,k}) = \left(P_{ind} + C_{eff}V^2\left(f_{op}^{i,k}\right)\right).T(t_i, vm_k) \tag{4.6}$$

and the total energy consumed for the application given by the sum of the energies of individual tasks in application and is calucted using the following equation.

$$E_{Total} = \sum_{i=1}^{n} E\left(f_{op}^{i,k}\right) \tag{4.7}$$

For simplicity, we considered only dynamic energy consumption in this work.

4.3.4 System Reliability

During the execution of an application, faults may occur due to hardware breakdown, software failures, cosmic ray radiation, etc. As the frequency of transient faults significant than permanent and intermittent faults [47] [102], we focused on transient faults in this work. Cloud provides shareable heterogeneous computing resources. The computing resource failures are inevitable and have conflicting effects on application performance and energy consumption. Processors failures are discrete events and assumed to follow a Poisson process [116]. The operating frequency of CPU influence the fault arrival rate λ [47]. This

fault arrival rate λ influences the performance of a node where a computation-intensive application running and hence the reliability of such node is essential.

We assume that the transient faults happen while individual tasks are in execution. However, with the effect of DVFS, the systems operating frequency can influence the error arrival rate and its corresponding supply voltage. Therefore, the fault rate is represented as in [102] is given in equation (4.8).

$$\lambda(f_{op}^k) = \lambda_0 g(f_{op}^k) \tag{4.8}$$

where λ_0 is the initial fault arrival rate at f_{max}^k, f_{op}^k is operating frequency of vm_k, $g(f_{op}^k)$ is a decreasing function and $g(f_{max}^k) = 1$. Generally, equation (4.12) is known as an exponential relation between the λ and the circuit's critical cost. In our scheduling approach and experimental analysis, we assume the model proposed in [102] and afterward used in [47] expressed as exponential model as in equation (4.9).

$$\lambda(f_{op}^k) = \lambda_0 g(f_{op}^k) = \lambda_0 10^{\frac{d(1-f_{op}^k)}{1-f_{min}^k}} \tag{4.9}$$

where λ_0, f_{op}^k and $g(f_{op}^k)$ are same as mentioned before. The positive constant d stands for the faulty rate dependency on frequency scaling and corresponding voltage. It can be easy to perceive exponential increase in λ with frequency scaling for energy saving, i.e. λ is maximum at the minimum allowed CPU frequency.

$$\lambda_{max} = \lambda_0 10^d, \quad \text{for} \quad f_{op}^k = f_{min}^k \tag{4.10}$$

Considering the transient fault model in [102] [47], which follows Poisson distribution model, the reliability R of a task t_i running on vm_k is calculated as follows:

$$R(f_{op}^{i,k}) = e^{-\lambda(f_{op}^k)\frac{T(t_i,vm_k)}{f_{op}^k}} \tag{4.11}$$

where $f_{op}^{i,k}$ is the operating frequency of the node where task t_i running. The reliability of application W with n number of tasks is the product of individual task reliability.

$$R_W = \prod_{i=1}^{n} R(f_{op}^{i,k}) \tag{4.12}$$

4.3.5 Problem Specification

The main aim of the proposed approach is to reduce energy utilization while maximizing system reliability. By considering the system models discussed in Sections 4.3.1 to 4.3.3 we have formulated the mathematical optimization specifications of the scheduling problem as follows:

$$\text{Energy: Minimize } (E_{Total})$$
$$\text{Reliability: Maximize } (R_W)$$

4.4 Algorithm implementation

The proposed EERS approach is capable of minimizing energy consumption and maximizing system reliability while meeting the user-defined deadline. It includes four sub-algorithms, such as task rank algorithm, task clustering algorithm, sub-target time distribution algorithm, and cluster-VM mapping algorithm, which reduce energy consumption and maximize the system reliability. This section presents an implementation of these sub-algorithms precisely.

4.4.1 Task rank calculation algorithm

Workflow tasks rank order is established in this stage to fulfill the requirement of task scheduling. The task ranks are established in such a way to meet the precedence constraints and finds a topological order for scheduling. To prioritize tasks in W without disturbing dependencies, each task t_i is assigned a rank $rank(t_i)$, that can be computed recursively staring with the exit task t_{exit} [47] as follows

Step-1: The exit tasks rank initialized to its average computing time

$$rank(t_{exit}) = \overline{T}(t_{exit}) \tag{4.13}$$

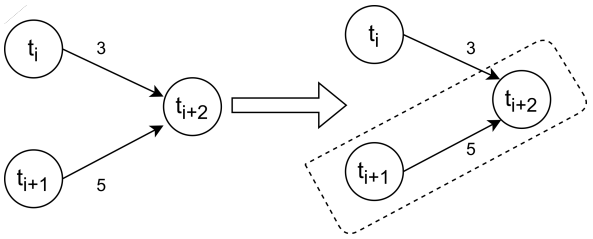

FIGURE 4.1: Clustering of Tasks

Step-2: For each task compute rank recursively according to the following expression

$$rank(t_i) = \overline{T}(t_i) + max_{t_j \in child(t_i)} rank(t_j) \tag{4.14}$$

where $\overline{T}(t_i)$ is the average computation time of t_i on different VMs. Estimate ranks of all tasks by repeating the above steps for each task in workflow.

4.4.2 Task clustering algorithm

Once the parent task completes execution, its generated output transfers to its child tasks. If both are scheduled on different VMs, then communication energy consumption incurs. A large amount of data communication energy is consumed during inter-processor communication. We can avoid this energy consumption by grouping the tasks and then schedule on the same machine. Consider three tasks t_i, t_{i+1} and t_{i+2} with the dependency shown in Figure 4.1, the task t_{i+2} has two parents, t_i and t_{i+1} with communication costs 3 and 5, respectively. Communication energy can be saved by grouping t_{i+1} and t_{i+2} to schedule on the same VM.

We adopt the clustering approach in [47] for the work proposed in this chapter and we extended it to minimize communication energy as follows:

Step-1: Staring from the entry task $t_i = t_{entry}$, for every task t_i, if task t_i not yet earmark for any cluster, then make a new cluster cl_l

Step-1a: add t_i to cl_l and sort the children of t_i

Step-2: For each child t_j of t_i, if cluster not been assigned and parents are assigned to a cluster

Step-2a: if a task t_j has more than one parent, then check its parents which transfers more data to it. Such parent can find as follows

$$t_k = max_{t_k \in parent(t_j)} w_{kj}, \tag{4.15}$$

if $EFT(t_j) \leq EST(t_{j-next})$ then $t_i \leftarrow t_k$, goto Step2

Step-2b: $t_i \leftarrow t_j$

Repeat Step 2 for the entire graph W, until each task assigned to some cluster.

4.4.3 Sub-Target Time Distribution algorithm

The target time to complete each task is based on the T_D is distributed to each task of the workflow. It is a proportionate increase in the effective execution time (makespan) of individual tasks and hence the application. This makespan extension can be done by reducing the frequency of the processor where it is running; hence, we can save energy. To implement sub-target time distribution we suggested a simple algorithm that can compute in polynomial time and steps for the sub-target time distribution algorithm are as follows

Step-1: Each task t_i in a given workflow, calculate the sub-target time using equation (4.16). The sub-target time is the deadline to complete execution of task t_i.

$$T_{sub-target}(t_i) = T(t_i) \frac{T_D}{minT_M} \tag{4.16}$$

where $T(t_i)$ is the effective execution time of task t_i, T_D and $minT_M$ are deadline and minimum makespan of application respectively.

Step-2: For each task t_i update EST and EFT using equations (4.4) and (4.5), respectively.

Every task is set with a new deadline or target completion time by repeating the above steps for each task in a workflow.

4.4.4 Cluster-VM mapping algorithm

Different from the clusters and utility grid computing environment, in a cloud environment, as long as the service available, then the cloud scheduling approach can provide a time slot to map the task to avail the service [38]. However, the cloud has inexhaustible resources, and it can offer VMs with different characteristics for users. But it is not always good to meet several optimization constraints. Therefore, in this section, we propose cluster-VM mapping to execute cluster tasks to maximize system reliability and minimize energy consumption. The steps to select a more appropriate VM for a cluster to execute its tasks as follows.

Step-1: For each task t_i (where $t_i \in cl_l$), calculate the optimal frequency for energy conservation on different available VMs using the equation (4.6).

Step-2: Calculate reliability of each task t_i (where $t_i \in cl_l$), on each VM using equation (4.11)

Step-3: Map the tasks of cluster cl_l to the highest reliable VM vm_k^{opt} to complete its execution; we denote it as $cl_l \rightarrow vm_k^{opt}$.

Step-4: If vm_k^{opt} is idle or if it executes in communication mode then scale down its operating frequency to its minimum i.e. $f_{op}^{i,k} = f_{min}^{i,k}$

We can map each cluster to the most suitable VM for energy conservation and maximize task reliability by repeating the above steps for all the clusters.

4.4.5 Slack algorithm

Consumption of electrical energy has developed into one of the primary interests of the cloud-data centers. In the context of workflow application scheduling, there is some idle time slots associated with VMs (slack time) while executing non-critical tasks. We can redeem this slack associated with non-critical tasks by scaling the supply voltage and frequency of the task, to conserve energy [38]. First, we need to estimate the latest start time (LST) of each task before we introduce the slacking algorithm. The LST of task t_i

$LST(t_i)$ is specified as follows

$$LST(t_i) = \begin{cases} T_D - T(t_{entry}) & \text{for } t_i = t_{exit} \\ min_{t_p \in child(t_i)}(LST(t_p) - T(t_p)) & \text{otherwise} \end{cases} \tag{4.17}$$

The slack time of task t_i is computed by

$$T_{slack}(t_i) = LST(t_i) - EST(t_i) \tag{4.18}$$

A critical task t_i has no room to reclaim i.e. $T_{slack}(t_i) = 0$, and for non-critical tasks $T_{slack}(t_i) > 0$. To reduce the frequency $f_{op}^{i,k}$ of a non critical task t_i to conserve energy, we used the following four steps:

Step-1: Calculate the slack time of the task t_i using equation (4.18) and mark all critical tasks as 'defined' as no idle slot to reclaim.

Step-2: Select a task t_i to change the frequency, which has the longest T_{path}, and its parents are marked 'defined', where T_{path} is the sum of the execution time of the tasks on the path from t_i to the 'defined' task.

Step-3: Reduce task frequency to extend the execution time and save energy as follows

$$f_{op}^{i,k} = \left(f_{max}^{i,k}\right) \frac{T_{path}}{T_{path} + T_{slack}(t_i)} \tag{4.19}$$

Step-4: Update execution time of t_i and mark it 'defined'.

Repeat the above steps for the entire DAG W until all tasks are marked 'defined'.

4.4.6 EERS Algorithm

We propose the EERS algorithm, which enables the cloud scheduler to get through less energy cost to complete an application and also maximizes system reliability while meeting the user-defined deadline. The EERS algorithm comprises five sub-algorithms which were

	The EERS Algorithm
1	BEGIN
2	Call the *Task rank calculation algorithm*
3	Call the *Task clustering algorithm*
4	Call the *Sub-Target Time Distribution algorithm*
5	**while** the workflow is not complete
6	Call the *Cluster-VM mapping algorithm*
7	**if** the task t_i is a non-critical task of workflow
8	**then** Call the *Slack algorithm*
9	**end if**
10	**end while**
11	END

FIGURE 4.2: The pseudo-code of the EERS algorithm

introduced in the above sections. We schedule a task to a specified VM when all of its predecessors finish i.e a task becomes schedulable when all of its predecessors complete their execution. When a current task completes execution then its successor tasks become schedulable. We specified a sub-deadline for every task before mapping it to the most appropriate VM. Thus, each task can be complete its execution within its target time, and the entire application can be completed within a specified deadline.

The pseudo-code for our EERS algorithm is presented in Figure 4.2. Firstly, the EERS algorithm calls the *task rank calculation* algorithm at line 2 in Figure 4.2, to realize a reasonable order of tasks to execute without loss of precedence constraints of workflow. At line 3 in Figure 4.2, the *task clustering* algorithm is called to minimize the communication cost which reduces energy consumption. Then *sub-target time distribution* algorithm for energy conservation by decreasing the task frequency while meeting user-defined quality parameter(deadline) in line 4. The *cluster-VM mapping* algorithm to select optimal VM to conserve energy and maximize task reliability is called at line 6 in Figure 4.2. Finally, a *slack algorithm* is called at line 7 in Figure 4.2 to reclaim the slack of non-critical tasks to further reduce the energy consumption. The clustering algorithm, task-VM mapping algorithm, and task slacking algorithm reduce energy consumption without compromising the performance.

TABLE 4.1: Simulation Environment Parameters

VM Parameter	Value(s)
CPU frequency level (f_{max})	2.0 GHz - 2.4 GHz
Computing capacity (MIPS)	1000 - 3000
RAM	512 MB
Bandwidth	1000 Mbps
Number of cores	1
Voltage supply (Vmax)	220 V

4.5 Performance evaluation

This section presents the performance of the EERS scheduling technique by conducting a series of simulation runs. We assume that the cloud data center has DVFS enabled virtual machines (VMs) and each VM has its computing resources, and bandwidth is constant between VMs instances. The computing performance of VMs and other simulation parameters present in Table 4.1 and are self-defined. The other parameter such as failure rates are 10^{-5} to 10^{-7} failures/s as in [47]. For ease, we suppose that the frequency of VM is directly proportional to its computing performance, which is realizable from the experimental point of view. Moreover, every VM operates at different levels of the frequency with minimum and maximum thresholds, and the DVFS takes advantage of these frequency levels to scale the task's operational frequency to conserve energy.

To simulate a cloud environment, the WorkflowSim tool is used. The strength of the EERS algorithm is evaluated using two real workflows: Montage and CyberShake. Both Montage and CyberShake applications have been represented as a workflow that can be executed in Grid environments and are mostly used benchmarking datasets for testing workflow optimization algorithms in clouds. The topological organization of these four scientific workflows is depicted in Figures 1.3 and 1.4 respectively.

The main focus of experimental results is on reliability and energy utilization. These will change with the varying workloads (number of tasks in workflow) and a varying number of VMs to execute the selected workload. To expose the strength of our proposed scheduling algorithm on reducing energy consumption and assured reliability the simulation results were compared with other existing works. We perform three well-known scheduling approaches HEFT [92], EES [63], and REEWS [47] to compare with our algorithm. The

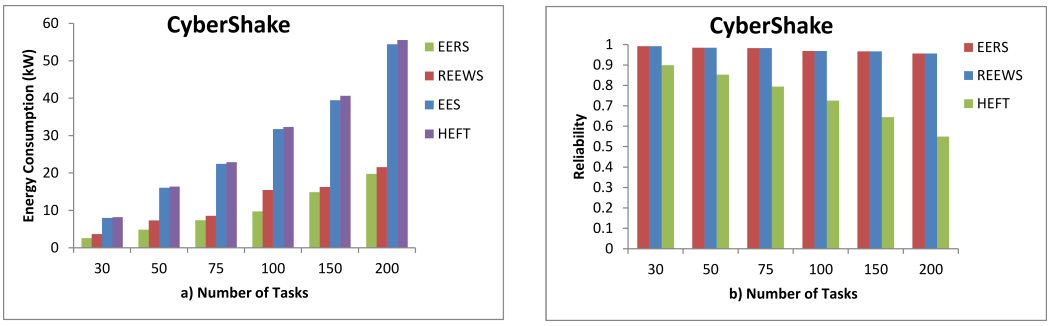

FIGURE 4.3: **a)** Energy consumption and **b)** Reliability for various workloads on CyberShake Workflow

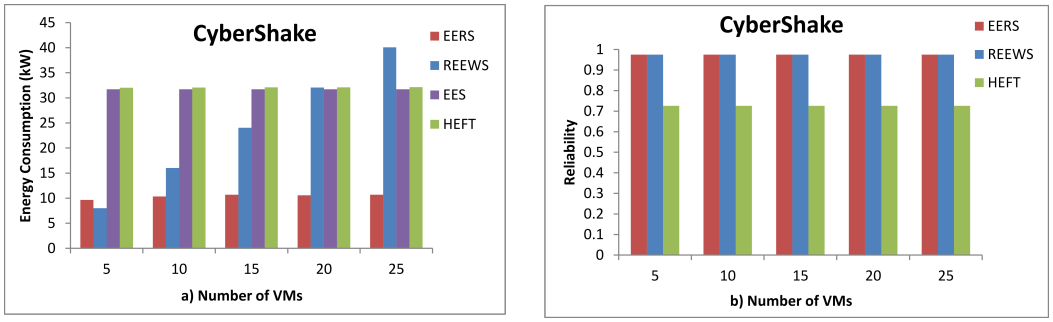

FIGURE 4.4: **a)** Energy consumption and **b)** Reliability for diffrent number of VMs on CyberShake Workflow

HEFT approach is a prominent list-based heuristic for workflow application scheduling to optimizing the makespan. In every step, HEFT chooses the highest priority (rank value) task to assign a processor, which reduces the EFT (earliest finish time) with an insertion-based approach. The EES (Enhanced Energy-efficient Scheduling) algorithm is a HEFT-based approach to conserve energy while meeting the quality parameters. The fundamental idea of the EES method exploits the slack time on non-critical tasks and globally allocates them to minimize energy. The REEWS is a heuristic algorithm to maximize the application's reliability and minimize energy consumption while meeting user-defined quality constraints. The REEWS algorithm works in four stages: 1) task priority calculation to preserve dependencies; 2) task clustering to minimize communication cost; 3) distribution of target time (user-defined deadline), and 4) mapping the cluster to VM with appropriate frequency/voltage levels.

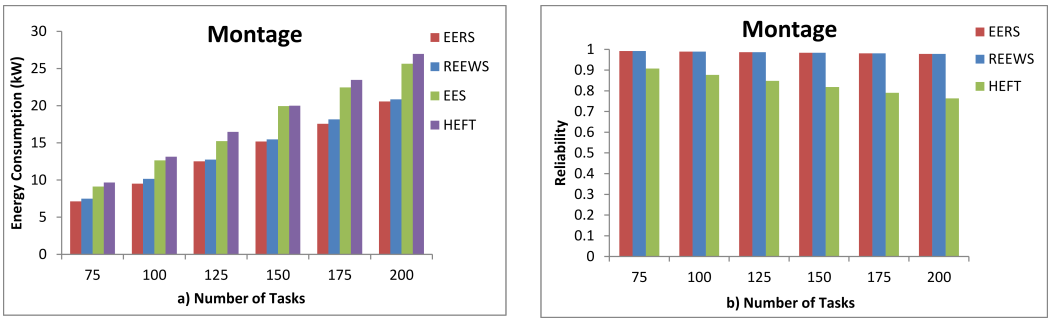

FIGURE 4.5: **a)** Energy consumption and **b)** Reliability for various workloads on Montage Workflow

4.5.1 Performance Evaluation with different workloads

First, we evaluated the performance of our algorithm for different workloads i.e by varying the number of tasks as 30, 50, 75, 100, 150, and 200 on *CyberShake* and 75, 100, 125, 150, 175, and 200 on *Montage* real-world scientific workflows. The simulation results for energy consumption and system reliability are depicted in Figures 4.3 and 4.4 respectively. In energy objectives, our proposed EERS consistently reduced energy concerning the workloads. It saved more energy as compared to the HEFT, EES, REEWS as the fact that the EERS is efficient in allocating resources to the tasks in such a way to reduce energy consumption. Moreover, EERS efficient in clustering the tasks to reduce the communication costs and hence saves energy. Finally, the EERS used a task reclaiming approach to take the advantage of the DVFS techniques to reduce energy utilization by lowering the task's supply voltage and frequency. The REEWS algorithm is an efficient technique in maximizing the system reliability which outperformed the state-of-the-art techniques such as RHEFT and PALS in reliability objective. Our proposed EERS gives better reliability with HEFT and on par with REEWS but in every case, our proposed EERS approach consumed less energy compared with the other approaches.

4.5.2 Performance Evaluation with different number of VMs

Further, we evaluated the performance of the EERS approach for different numbers of VMs on both CyberShake and Montage workflows. We considered 5, 10, 15, 20, and 25 VMs on CyberShake and 15, 18, 20, 24, and 28 VMs on Montage. The simulation results for energy consumption and system reliability on CyberShake and Montage workflows are

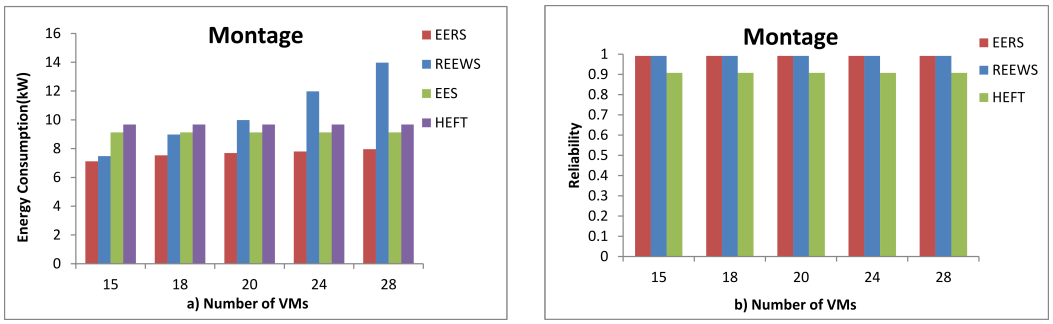

FIGURE 4.6: **a)** Energy consumption and **b)** Reliability for diffrent number of VMs on Montage Workflow

depicted in Figures 4.5 and 4.6 respectively. Based on energy consumption and reliability the EERS algorithm efficient in selecting the number of VMs and as well as the type of VMs. Hence, in this case, our EERS algorithm saved more energy on both workflows compared to other approaches. With varying numbers of processors/VMs also our approach maintained good reliability this is because the EERS algorithm considers failure rate before selecting resources for mapping tasks to that resources.

4.6 Conclusion

Recently, the need for energy conservation and maximizing system reliability has become important research. In this work, we considered scientific real-world workflows to schedule in the cloud environment. We present Energy Efficient and Reliability-Aware Scheduler (EERS) for scheduling workflow tasks in Cloud Computing to minimize energy consumption and maximize the application reliability while satisfying user-defined deadline constraints. Our proposed EERS approach comprises of five sub-algorithms i) task rank calculation algorithm, ii) task clustering algorithm, iii) Sub-target time distribution algorithm iv) cluster-VM mapping algorithm, and v) slack algorithm. We discussed our EERS performance in Section 5 by performed considerable simulation runs on the WorkflowSim toolkit and evaluated our algorithm on real-world scientific workflows CyberShake and montage for different numbers of virtual machines and different workloads. We compared the performance of our EERS algorithm with popular workflow scheduling techniques such as HEFT, EES, and REEWS. It was observed from the simulation experimental results that our proposed approach consumed less energy with

maximizing the system reliability in all the cases. We can conclude that the time complexity of the proposed sub-algorithms is polynomial. Simulation experiments' outcomes reveal that our EERS approach surpasses other algorithms in both energy consumption and reliability. It showed 37.46%, 19.75%, and 26.91% energy saving with Montage workload and 23%, 19.68% and 200% energy saving with CyberShake workload compared to REEWS, EES, and HEFT algorithms respectively.