Chapter 3

A heuristic-based energy-efficient and cost-aware scheduling algorithm with deadline constraints

This chapter[1] proposes a deadline constraint-based scheduling algorithm called ECWS for workflows in a cloud environment. The ECWS is HEFT based heuristic for cloud schedulers that address different objectives such as energy utilization, monitoring cost, and resource utilization. It comprises three sub-algorithms such as ratio of effectiveness (RE) Calculation, RE threshold selection, and slack algorithm. The efficiency of the algorithm is evaluated using WorkflowSim toolkit and four different scientific workloads like CyberShake, Montage, SIPHT, and LIGO. The simulation experimental results show that it outperformed the related well-known algorithms in terms of reducing energy utilization, cost savings, and maximizing resource utilization.

[1]This chapter is derived from: Medara, Rambabu, Ravi Shankar Singh, and Mahesh Sompalli. "Energy and cost aware workflow scheduling in clouds with deadline constraint." Concurrency and Computation: Practice and Experience: e6922.

3.1 Introduction

Workflow is regarded as a set of computing tasks that are handled in a sequence to realize real-world applications [82]. It is a familiar and effective way of modeling various scientific problems in parallel and distributed systems [38]. DAG model is common for scheduling workflows in a cloud environment. A DAG represents the characteristics of workflow, which include task execution times, the amount of data communicated between tasks, and the dependencies among the tasks. The scientific workflows are computationally intensive, data-intensive, and hence usually take many hours to execute. Cloud technology offers various benefits for scientific applications over traditional computing such as rapid resource provisioning, elasticity to scale resources dynamically, collaboration efficiency, and cost-saving. To take advantage of the cost-effectiveness and scaling flexibility of cloud computing it is essential to perform computational-intensive workflows on the cloud [83].

Cloud computing has been better known for its flexibility, reliability, and security. It has become a leading technology industry and has seen a massive expansion in the last decade. Due to this, some leading global tech giants have created large cloud data centers across the globe for facilitating cloud services. These datacenters use diverse sets of advanced machinery and equipment to function, which in turn requires enormous energy in the form of electricity. The study of the energy problem in clouds has become one of the major challenging issues. In the wake of the persistent growth rate for cloud-based services, there has been substantial demand for energy utilization in cloud data centers. As a consequence, an increase in cloud operating costs and as well carbon discharge into the environment [84].

Various benefits of using cloud technologies, such as rapid elasticity, scalability, availability, ubiquitous access, and reliability, have made the cloud an attractive stand-in for conventional Information Technology (IT) infrastructures [85]. Cloud computing has developed as a favorable computing model for public and private research institutes and industries to deal with the ever-growing demand for computing and storage.

Cloud resources support the execution of computationally intensive workflow applications. The workflow scheduling in the cloud environment is a well-known NP-complete problem [34]. There is no capping on the amount of cloud resources access to schedule workflow applications. Moreover, the cloud providers charge users based on the leased time unit rather than the actual resource used for computing. Therefore, users need to pay for the whole leased time unit. Most of the cloud vendors provide different types of virtual machines (VMs) at different prices. Therefore, to take advantage of these pricing models, efficient workflow task scheduling is required for maximal resource utilization during the leased period.

This chapter introduces an energy and cost-aware workflow scheduling (ECWS) approach for cloud schedulers to reduce energy utilization and the execution cost of the scientific workflows.

The rest of this chapter is structured as follows. Section 3.2 presents the related work and the proposed system models are described in Section 3.3. Then Section 3.4 gives the four sub-algorithms of the proposed ECWS algorithm. The experimental setup and results analysis demonstrated in Section 3.5. In the end, the conclusion and future scope of work are addressed in Section 3.6.

3.2 Related work

Over the last few years, researchers have been paying considerable attention to energy-aware cloud resource management and establishing models to contribute to workflow task scheduling. Workflow scheduling efficiency and effective cloud resource management rely on different factors including user request arrival rate, accessibility and reliability of the data center resources, and the workloads of the tasks.

Many of the recent works for workflow task scheduling in cloud computing considered different performance objectives like schedule length, VM utilization, and cost. A hybrid multi-objective scheduling algorithm in [86] addresses the cost and makespan of the application by considering the budget and workflow deadline constraints. It uses a recently introduced spider monkey optimization algorithm with a budget and deadline constraints approach. A two-phase list-based approach investigated in [87] to address various QoS parameters but it is a simple heuristic to trapped into local optima. Another multi-objective algorithm [88] for makespan and cost used an improved PSO. A relative distance-based VM allocation algorithm [89] which effectively improved both VM utilization and makespan. Another cost-effective approach in [90] schedules tasks from multiple workflows to effectively utilize the idle time slots of VMs for saving cost. Deng, Kefeng, et al. [91] proposed an efficient scheduler for scientific workflow applications that group similar tasks and datasets. It used a data staging mechanism to avoid tasks overlapping. A list-based heuristic called HEFT [92] approach gives the best makespan. A mathematical scheduling model proposed by [93] optimizes the cost of workflows while satisfying a deadline constraint. A HEFT-based multi-objective approach (MOHEFT) developed by Durillo and Prodan [94] computes a set of Pareto-based solutions. Another multi-objective approach proposed in [95] addressed the multiple conflicting objectives such as server workload, schedule length, load-balancing, and reliability using the firefly algorithm. Some evolutionary algorithms effectively reduce the cost and schedule length. Tsai et. al [96] have combined the Taguchi method with the Differential Evolution algorithm resulting in the IDEA algorithm. This algorithm is well balanced on both exploration and exploitation. It is examined that these works ignored the energy objective.

Reducing the energy consumption of data centers is a major concern these days. Numerous competent scheduling approaches to minimize energy consumption have been researched. It proposes task clustering and slacking approaches for energy saving. An energy-aware heuristic in [46] addresses energy, monetary cost, and quality of service objectives. The above two works not minimizing the number of active machines. A HEFT-based slack room reclaiming algorithm called an enhanced energy-efficient scheduling (EES) technique is proposed to minimize the energy consumption while preserving the workflow makespan [63] but it only used slack algorithm to reduce energy consumption. A Dynamic Voltage and Frequency Scaling (DVFS) technique was used by Cao and Zhu [37] to scale down the CPU operating frequencies of VMs considering the workflow deadline. However, they ignore the cloud datacenter pricing models as the VM instances are charged by an hourly-based pricing model. Two list-based energy-aware task scheduling algorithms proposed in [61], called the Enhancing HEFT (EHEFT) and the Enhancing Critical Path on a Processor, addresses the energy-efficient and schedule length in workflow scheduling. But these two approaches, only find power inefficient processors to reduce energy consumption. A multi-objective approach to minimize the energy consumption of the application proposed in [97] uses the Genetic Algorithm (GA) and a gap scheduling is used to maximize the resource utilization. Although, most of the works reduce energy consumption but are not considered the monetary cost objective.

3.3 Problem Modeling

Our problem comprises scheduling the scientific workflow applications in clouds in a manner that the energy consumption and monetary cost are reduced while meeting the user-specified deadline. This section describes the system models such as workflow model, cloud datacenter model, cost model, and energy model used in the proposed approach.

3.3.1 Datacenter Model

We consider cloud datacenter with k types of virtual machines $VM = \{vm_1, vm_2, vm_3, \ldots.., vm_{k-1}, vm_k\}$. These VMs are charged based on time unit (hourly pricing model). Using the fact that the recent processors support DVFS techniques, that enable every processor to scale the voltage and frequency at runtime to reduce power utilization [98], we consider that all VMs in the cloud data center support DVFS. For example, Amazon EC2 A1 provides six types of VM instances [99]: a1.medium, a1.large, a1.xlarge, a1.2xlarge, a1.4xlarge and a1.metal. Each VM considered in this work is assumed to run at different levels of frequencies $(f_1, f_2, \ldots.., f_{l-1}, f_l)$. The switching time between frequencies roughly between 10 and $150\mu m$ which is insignificant. The computing resources (VMs) are characterized by computing capacity in a million instructions per second (MIPS), bandwidth (BW), etc. Therefore, the cloud scheduler has the potential in choosing the VMs to run the tasks to satisfy workflow constraints. A k^{th} VM operating at some level of frequency (op) denote as f^k_{op}.

3.3.2 Application Model

It is common to model a workflow W with precedence constraints among a set of tasks $Tw = \{t_1, t_2, \ldots., t_{n-1}, t_n\}$ as Directed Acyclic Graph (DAG). A DAG $W = (T_w, C)$ where C is the set of communication arcs between tasks. An arc between t_i and t_j $(t_i, t_j \in T_w)$ represents the dependency and is denoted as c_{ij} $(i, j \in C \; and \; i \neq j)$ where t_i is predecessor of t_j and t_j is immediate successor of t_i. A task with zero predecessors is an entry point t_{entry} and a task with zero successors is an exit point t_{exit}. If all required computing resources are allotted for a task then it prompts to execute. After finishing the t_i its output data transferred (in MB) to its immediate successors and it is marked as the edge weight between t_i and t_j as $w(c_{ij})$. The total execution time of W is denoted as T_M and deadline as T_D. The deadline, T_D is a user-specified time constraint for workflow applications. The data transfer time between t_i and t_j is $T(t_{ij})$ and is calculated using equation (3.1).

$$T(t_{ij}) = \frac{w(c_{ij})}{BW} \tag{3.1}$$

where BW and $w(c_{ij})$ are bandwidth and amount of data tranffered from t_i to t_j respectively. The effective execution time of any task includes its execution cost and data communicated to its immediate successors. For eaxapmle, the effective execution time of t_i is on k^{th} virtual machine is $T(t_i, vm_k)$ and is estimated as in follows equation:

$$T(t_i, vm_k) = \frac{w_i}{f_{max}^k} + T(t_{ij}) \tag{3.2}$$

In the context of scheduling analysis, it is common to looks for the latest and earliest points of the task's ending and beginning. The earliest start time (*EST*) and earliest finish time (*EFT*) of task t_i are measured using equations (3.3) and (3.4) respectively.

$$EST(t_i) = \begin{cases} 0 & \text{if } t_i = t_{entry} \\ max_{t_p \in parent(t_i)} EFT(t_p) & \text{otherwise} \end{cases} \tag{3.3}$$

$$EFT(t_i) = EST(t_i) + T(t_i, vm_k) \tag{3.4}$$

The minimum schedule length is the earliest completing point of the exit task of the workflow $EFT(t_{exit})$ i.e., $minT_M = EFT(t_{exit})$. Without loss of assumption, the user-specified deadline for completing the workflow must be not less than the minimum makespan $T_D \geq minT_M$.

3.3.3 Cost Model

We consider a pay-as-you-go cost model where the VM instances are being leased for the applications in an hourly-based manner. So we have defined the cost of all the VMs available on an hourly basis. Hence, the monetary cost of any task executing on a leased machine is simply the hourly price of that VM instance. Hence the overall cost of application $Cost(W)$ is the sum of the costs of all the tasks.

3.3.4 Energy Model

The cloud server's power consumption results from different components such as CPU, memory, network interfaces, storage disks, and other underlying circuits. Among these, the CPU is the major contributor. While executing workflow applications in a cloud platform we can see three components of power consumption that include independent, dynamic, and static. The independent power utilization P_{ind} is free from the processor frequency and voltage whereas the dynamic power consumption $P_{Dynamic}$ depends on the processor's supply frequency. Different components such as memory, storage, input-output devices, etc., come under independent power utilization and this component can be minimized by keeping the system in sleep mode [100]. The static power utilization P_{Static} is to remain the clock running, to maintain the basic circuits, etc.. and it can be avoided by shutting down the circuit. The power utilization is estimated by the product of the supply voltage (V) and current (I) of the circuit as in equation (3.5).

$$P_{Static} = V * I \tag{3.5}$$

$P_{Dynamic}$ is the dominant among the three components of power consumption. It is expressed as the sum of the transient capacitive-load power utilization [101]. We used the power model to estimate $P_{Dynamic}$ suggested [102]

$$P_{Dynamic} = C_{eff}V^2 f = C_{eff} f^3 \tag{3.6}$$

where C_{eff} is the effective-load capacitance, and f and V are frequency and voltage respectively. A CPU with clock speed (frequency) of f and supply voltage V, f is directly proportional to V. To emphasize the cloud server power consumption by considering both $P_{Dynamic}$ and P_{Static} power components, this work considers the power model used in [103]. The server power consumption when it is active (h=1) is calculated using equation (3.7) and it can be rewriting as in equation (3.8).

$$P = P_{Static} + h(P_{ind} + P_{Dynamic}) \tag{3.7}$$

$$P = P_{Static} + h(P_{ind} + C_{eff}V^2 f) \tag{3.8}$$

Note h is set to zero to specify that the server is not in an active state. Because the $P_{Dynamic}$ component is the most compelling, hence the other components are not considered in this work. Further, the energy consumption of any active server is a product of power utilization and service time as in equation (3.9).

$$E = P * t \tag{3.9}$$

The energy utilization of any task t_i on vm_k calculated using equation (3.10).

$$E(f_{op}^{i,k}) = \left(P_{ind} + C_{eff}\left(f_{op}^{i,k}\right)^3\right).T(t_i, vm_k) \tag{3.10}$$

The overall energy utilization for the workflow execution is the summation of energies of all tasks in W and it can be measured using equation (3.11).

$$E_{Total} = \sum_{i=1}^{n} E\left(f_{op}^{i,k}\right) \tag{3.11}$$

3.3.5 Scheduling Model

Considering the system models discussed in the above Sections 3.3.1 to 3.3.4 the workflow scheduling problem deal with many conflicting optimization objectives such as cost, energy, and resource utilization can be formulated as mathematical models as follows:

Cost: Minimize $Cost(W)$
Energy: Minimize E_{Total}
Resource utilization: Maximize $Util(\%)$

Note: the resource utilization model $Util(\%)$ discussed in Section 3.4.3.

3.4 Proposed Methodology

The proposed ECWS algorithm is efficient in minimizing energy consumption and cost by considering the user-specified deadline constraint. The ECWS runs in four phases (sub-algorithms), such as initial task scheduling with the HEFT algorithm, identification of inefficient processors by evaluating the RE metric, removal of identified inefficient processors by selecting an ideal RE threshold to minimize the monetary cost and energy consumption. Finally, task slacking algorithm to reclaim slack period associated with non-critical tasks for further energy saving. In this section, we introduce the implementations of these phases precisely as follows.

3.4.1 Task Scheduling with HEFT algorithm

Initially, the workflow tasks are scheduled using HEFT [92], which is an efficient approach to get a short schedule length. The HEFT is a well-known list-based heuristic scheduling algorithm for minimizing the schedule length in workflow applications. The HEFT algorithm runs in two phases, such as task prioritization to calculate the upward rank of tasks and processor selection to map the task to VM based on upward rank, which optimizes EFT using an insertion-based approach. We calculated energy consumed E_{heft} using equation (3.11) and we used it in Algorithm 2 as baseline energy. A workflow with n number of tasks, the HEFT algorithm has time complexity $O(n^2 + vn)$, where v is the number of processors (VMs).

3.4.2 Calculate Ratio of Effectiveness (RE) Values

Generally, the cloud is viewed as virtually boundless resources that could provision on-demand, however, the cloud data center offers different virtual machine instances to the users for scheduling tasks. In this work, we considered a bounded number of heterogeneous VMs as in [92] for processing the application. But these VM instances, always not suitable to process tasks, mainly when a task wastes a considerable amount of its leased time. As the scientific workflows are very big, they usually demand more computing hours. It is essential to select the optimal VM to lower cost and energy utilization. In this phase, we identify the inefficient processors that consume energy and incur monetary charges while sitting idle most of the time. A processor is marked as inefficient based on the metric, called Ratio of Effectiveness (RE) defined between any two processors. The metric RE is estimated based on the mutual/common time between

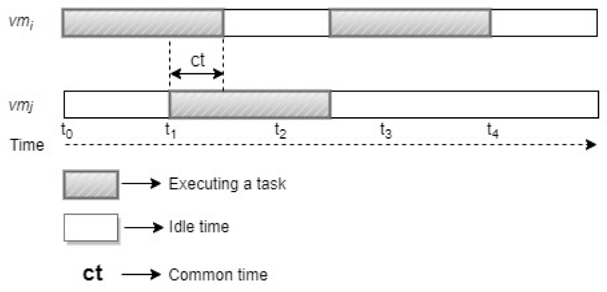

FIGURE 3.1: Mutual time between two VMs.

any two tasks. The mutual time between two processors vm_i and vm_j is the total amount of time both VMs are executing simultaneously as shown in Figure 3.1. It is clear that except for common time (ct), vm_j is idle while vm_i is busy processing a task and vice versa. Hence removing one of these VMs from the set of specified VMs v makes a little bit of a difference in makespan which is not significant but we can save energy and cost. The mutual time can be calculated by comparing every event happening on vm_i with every other event taking place on vm_j. Occurrence of an event refers to the execution period of a task, hence it is associated with a start and finish times of a task. Let the events $\{e_1, e_2, \ldots, e_n\}$ take place on vm_i. Without loss of generality assume event e_k takes place before event e_j for all $j > k$. And assume schedules on vm_i $skd[vm_i]$ and vm_j $skd[vm_j]$ contain set of events, say $\{e_1 U e_2 U \ldots U e_n\}$. Then the mutual/common time between any two VMs calculated as follows:

Read the schedules of two VMs say vm_i and vm_j. For each event e_i happening on vm_i and e_j happening on vm_j, the common time between these events starts when either e_i starts while e_j is running or e_j starts when e_i is running. And the common time ends when one of the events finishes its execution. The common time between any two events e_i and e_j estimated using one of the two cases as follows:

Case 1:

If $ST(e_i) \geq ST(e_j)$ and $ST(e_i) < FT(e_j)$ **then**
$commonT(e_i, e_j) = min(FT(e_i), FT(e_j)) - ST(e_i)$

Case 2:

If $ST(e_j) \geq ST(e_i)$ and $ST(e_j) < FT(e_i)$ **then**
$commonT(e_i, e_j) = min(FT(e_i), FT(e_j)) - ST(e_j)$

where $ST(e_i)$ and $FT(e_i)$ are start time and finish times of event e_i respectively. Similarly, $ST(e_j)$ and $FT(e_j)$ are start time and finish times of event e_j respectively. In the first case, *Case 1:* event e_i started while event e_j was executing and common time between these two events lasts till either e_i finishes or e_j finishes. The second case, *Case 2:* event e_j started while event e_i was executing and common time between these two events lasts till either e_j finishes or e_i finishes. The RE of any two processors (vm_i, vm_j) is denoted as RE_{ij}, and it is calculated as in equation (3.12).

$$RE_{ij} = \sum_{e_i \in skd[vm_i] \text{ and } e_j \in skd[vm_j]} \frac{commonT(e_i, e_j)}{makespan} \tag{3.12}$$

where $commonT(e_i, e_j)$ is the common time betwen events (e_i, e_j), *makespan* is the total schedule length, and $skd[vm_i]$ and $skd[vm_j]$ are schedules on vm_i and vm_j respectively. The ratio of effectiveness between any two processors can be calculated to prepare the set of REs using the pseudo code in Algorithm 1.

Algorithm 1 RE Calculation

```
1: procedure RE_CALCULATION($v$)
2:     for $Each\ vm_i$ do
3:         for $Each\ vm_j$ do   $i \neq j$
4:             Calculate $RE(vm_i, vm_j)$ using equation (12)
5:         end for
6:     end for
7: end procedure
```

We assume t_i number of tasks scheduled on vm_i, similarly t_j number of tasks on vm_j. Hence the common time can be estimated in $O(t_i + t_j)$, where $i, j = 1, 2, 3 \ldots, n$. Hence, the time complexity of the RE calculation approach is $O(vn + vn)$ or simply $O(vn)$, where n is the number of tasks and v is the number of virtual machines.

3.4.3 Selecting RE Threshold

In this phase, ideal RE threshold values are calculated dynamically based on the workflow deadline to switch off inefficient VMs. By removing the inefficient VMs we can minimize energy consumption and cost. A threshold value is selected so that after switching off the inefficient VM(s) the makespan of workflow should be less than or equal to the specified deadline i.e. $T_M \leq T_D$. It provides the best energy value than other threshold values that satisfy the condition $T_M \leq T_D$. All the VMs with RE value less than the threshold are inefficient in task execution and consume most of the energy by sitting idle. By switching off such inefficient VMs we can save energy and cost. We chose inefficient VMs to remove from the VMs list based on RE values and power efficiency.

The power efficiency of vm_i is the ratio of its MIPS to its voltage. For all the VM pairs with RE value less than the threshold, a VM is removed from each pair based on power efficiency. An optimal threshold saves energy consumption while makespan is less than the deadline. If the chosen deadline is never met the condition ($T_M \leq T_D$) then it uses all VMs to give the best makespan.

Algorithm 2 gives the pseudo-code for the RE threshold selection that first runs the application with the HEFT scheduler using a set of the specified VMs v. The HEFT scheduler does not consider the power inefficiency of the processors to minimize energy. Thus, removing some inefficient processors vm_{inef} there is the possibility of minimizing energy consumption while meeting the application deadline. So, Algorithms 2 calls the RE calculation algorithm and removes some inefficient VMs based on RE threshold and power efficiency. The RE threshold value is set dynamically so that it can minimize the number of active processors to save energy without failing the workflow's deadline constraint. To select an ideal RE threshold value we consider threshold in increments of small value δ. The smaller the δ precise the value of the threshold is, and then schedule the application again with HEFT scheduler on v' processors. Where v' is defined as in equation (3.13).

$$v' = v - vm_{inef} \tag{3.13}$$

where vm_{inef} is the number of identified inefficient VMs. After rescheduling the application with v' processors if the schedule length is less than the deadline then Algorithm 2 returns a new set of VMs v' otherwise an initial set of VMs v is used for scheduling the application.

After removing the inefficient processors using threshold selected in the Algorithm 2, the scheduler maps the tasks to the new set of v' VMs and saves energy by maximizing the CPU resource utilization. The CPU resource utilization of a processor or a vm_i is calculated

Algorithm 2 RE Threshold selection

```
1: procedure RE_THRESHOLD_SELECTION($W$, $T_D$, $v$)
2:     $rm\_vms = 0$;
3:     $re\_threshold = 0$;
4:     $\delta = 0.01$;
5:     $best\_energy = E_{heft}$;
6:     while ($re\_threshold \leq 1$) do
7:         HEFT_SCHEDULE();
8:         RE_CALCULATION();
9:         Remove power inefficient vms;
10:        rm_vms= removed vms;
11:        Update new set of VMs $v' = v - rm\_vms$;
12:        Reschedule W with HEFT_Scheduler using new
           set of VMs v';
13:        $T_M = getMakespan()$;
14:        Calculate energy using Equation (3.11);
15:        if ($T_M \leq T_D$ and $energy < best\_energy$) then
16:            $best\_energy = energy$;
17:        end if
18:        $re\_threshold += \delta$;
19:    end while
20:    Return $v'$
21: end procedure
```

using the equation (3.14).

$$Util(vm_i) = \frac{T_{active}(vm_i)}{T_{active}(vm_i) + T_{idle}(vm_i)} \tag{3.14}$$

where $T_{active}(vm_i)$ and $T_{idle}(vm_i)$ are the active time and the idle time of vm_i respectively. The overall percentage of resource utilization for the application calculated using equation (3.15).

$$Util(\%) = \sum_{i \in v'} \frac{T_{active}(vm_i)}{makespan * v'} * 100 \tag{3.15}$$

where v' is the reduced number of VMs using RE threshold values. The Algorithm 2 removes possible inefficient VMs based on RE values, hence the time complexity depends on the number of VMs v and the number of tasks n. In 6^{th} and 7^{th} steps, we are calling HEFT schedule twice and RE calculation algorithms which have time complexity $O(n^2 + vn)$ and $O(vn)$ respectively. In step 8 removes inefficient processors, for this purpose pick a pair of VMs with least RE values in $O(v^2)$ time. The time complexity for removing one of the VMs from this pair based on power efficiency is $O(n^2+vn+v^2)$. Since we check for all VMs, we keep removing one vm at a time. As there is a set of v VMs, the total time complexity for Algorithm 2 is $O((n^2+vn+v^2)v)$.

3.4.4 Slack reclaiming

Attention needs to pay to the energy consumption in cloud data centers. Many works [38] [63] have successfully proven that energy consumption can be reduced by reclaiming the slack time of VMs in connection with workflow task scheduling while executing non-critical tasks. By reclaiming the slack connected with such tasks significant energy can be saved. A popular method called the DVFS technique is used to reclaim such slack by lowering the supply frequency and voltage of the VMs where non-critical tasks are running [38] [63]. To estimate slack time connected with any task we calculated LST (latest start time). The LST of task t_i $LST(t_i)$ is estimated as in equation (3.16) and then slack of any task t_i using equation (3.17).

$$LST(t_i) = \begin{cases} T_D - T(t_{entry}) & \text{for } t_i = t_{exit} \\ min_{t_p \in child(t_i)}(LST(t_p) - T(t_p)) & \text{otherwise} \end{cases} \tag{3.16}$$

$$T_{slack}(t_i) = LST(t_i) - EST(t_i) \tag{3.17}$$

There is no idle time periods associated with critical task t_i to reclaim it i.e. $T_{slack}(t_i) = 0$. But in the case of non-critical tasks, the slack must be non zero $T_{slack}(t_i) > 0$. To reclaim the slack period, the execution time of a task t_i extended without affecting the EST of its child tasks. This can be achieved by reducing the frequency and voltage of task t_i. The entire process is called slack reclaiming which further reduces the overall energy utilization of the application without compromising the performance. The pseudo-code for a slack algorithm is given in Algorithm 3. It calculates the LST and slack of each task t_i using Equations (3.16) and (3.17) respectively and if slack is associated with any task then a new deadline $T_{sub-deadline}(t_i)$ set to that task by considering its successors EST. A new deadline is calculated for a task t_i using equation (3.18) if t_i and its successors are scheduled on the same processor. In the case of t_i and its successors scheduled on different processors then a new deadline of t_i is set using Equation (3.19).

$$T_{sub-deadline}(t_i) = max(LST(t_i), EST_{j \in successor(t_i)}(t_j)) \tag{3.18}$$

$$T_{sub-deadline}(t_i) = max(LST(t_i), min(EST_{j \in successor(t_i)}(t_j), EST(t_{i-next}))) \tag{3.19}$$

where $EST(t_{i-next})$ is the earliest start time of the next task to the task t_i scheduled on processor vm_i. Processor operating frequency and voltage are reduced to extend the task execution time using equations (3.20) and (3.21) respectively.

$$f_{op}^{i,k} = \left(f_{max}^{i,k}\right) \frac{T(t_i, vm_k)}{T_{sub\_deadline}(t_i) + T(t_i, vm_k) - EFT(t_i)} \tag{3.20}$$

$$V_{op}^{i,k} = \left(V_{max}^{i,k}\right) \frac{T(t_i, vm_k)}{T_{sub\_deadline}(t_i) + T(t_i, vm_k) - EFT(t_i)} \tag{3.21}$$

where $f_{op}^{i,k}$ and $V_{op}^{i,k}$ are the operating frequency and voltage respectively to execute task t_i on virtual machine vm_k, and $T_{sub\_deadline}(t_i)$ is the new deadline of the task t_i after extension of task execution time and is estimated as in Algorithm 3.

The time complexity of the slack algorithm depends on searching the tasks on the graphs. The worst time complexity for searching tasks on the graph takes $O(n^2)$, where n is the number of vertices in a graph. Hence the worst-case time complexity of the slack algorithm is $O(n^2)$.

3.4.5 ECWS Algorithm

We present the ECWS algorithm which runs in four phases introduce in the above sections. The ECWS algorithm reduces the energy consumption and also enables the cloud scheduler to disburses fewer prices to finish a workflow. The proposed algorithm completes workflow such that the execution time must not exceed the defined deadline. Our algorithm maps each task t_i to an appropriate VM type. Even the task extension in the slacking algorithm to reclaim the slack time does not affect the overall makespan as we have specified a sub-deadline for each task. Consequently, each task can be completed within its sub-deadline. Therefore, the entire workflow will be finished on time. Algorithm 4 gives the pseudo-code for the ECWS approach.

Algorithm 3 Slack algorithm

```
 1: procedure SlackAlgorithm($W$, $T_D$, $v'$)
 2:     Calculate LST and Slack of each task $t_i$ using
        equations (3.15) and (3.16) respectively;
 3:     while $(T_{Slack}(t_i) > 0)$ do
 4:         for each virtual machine $vm_i$ do
 5:             for each task $t_i$ on $vm_i$ do
 6:                 $T_{sub-deadline}(t_i) = LST(t_i)$;
 7:                 for all child tasks of $t_i$ do
 8:                     Set new deadline for task $t_i$ using
                        Equation (3.17);
 9:                 end for
10:                 if the next task $t_j$ to be schedule on $vm_i$
                    is not the successor of the current
                    task then
11:                     Set new deadline for task $t_i$ using
                        Equation (3.18);
12:                 end if
13:                 Update frequency and voltage of task $t_i$
                    using equations (3.19) and (3.20)
                    respectively;
14:                 Update execution times of task $t_i$;
15:             end for
16:         end for
17:     end while
18: end procedure
```

Algorithm 4 The ECWS Algorithm

```
 1: procedure ECWS Algorithm($W$, $T_D$, $v$)
 2:     HEFT_Schedule();
 3:     RE_Calculation();
 4:     RE_Threshold_Selection();
 5:     while the W is not complete do
 6:         if a task has a slack i.e. $T_{slack}(t_i) > 0$ then
 7:             SlackingAlgorithm();
 8:         end if
 9:         Calculate resoureces utilization using
            Equation (3.14);
10:     end while
11: end procedure
```

TABLE 3.1: The ECWS algorithm summary

	RE Calculation	RE Threshold Selection	Slack Algorithm
Minimize cost	-	Yes	-
Minimize energy	-	Yes	Yes
Maximize resources utilization	-	Yes	-
Time complexity	$O(vn)$	$O((n^2+vn+v^2)v)$	$O(n^2)$

We consider the first task t_{entry} as a schedulable task and map the schedulable tasks to the specified efficient VMs. Once completed the currently executing task then its successors (child) may become the schedulable tasks. The process repeats until the completion of the entire workflow W. The functions of each sub-algorithm used in the ECWS algorithm along with their time complexities shown in Table 3.1. We find that each algorithm has polynomial time complexity. We conclude that the RE threshold selection and slack algorithms reduced significant energy consumption. The RE Threshold selection algorithm reduced the monetary cost and maximized resource utilization.

3.5 Performance Evaluation

The efficiency of the ECWS method was evaluated by conducting a series of experiments to study energy utilization, cost savings, and resource utilization. This work presumes that the cloud resources (VMs) have DVFS enabled and each VM has attributed different resources to accomplish task execution as discussed in section 3.3.1. Table 3.2 lists the VM instance specifications such as cost per hour, computing capacity, and frequency range. For ease, we suppose the processor's frequency is directly proportional to its computing discharge [38], and it is convincing from the experimental perspective. Further, each processor runs at various levels of frequency by fixing the lower and upper threshold values. We can get benefit from these levels of frequencies in reducing energy utilization by using the DVFS technique, which scales the running frequency of tasks for energy-saving.

For the purpose of simulating cloud environment, the WorkflowSim tool is used in this work, which is a toolkit that enables the users to model and simulates Infrastructure-as-a-Service (IaaS) cloud. The IaaS cloud provides virtualized computing resources (VMs) to perform workflow applications. To evaluate the performance of the ECWS approach we have

TABLE 3.2: VM parameters

Type	Cost ($/h)	Computing capacity (MIPS)	CPU frequency (GHz)	
			Minimum	Maximum
1	0.10	1000	0.50	1.00
2	0.20	1500	0.75	1.50
3	0.32	2000	1.00	2.00
4	0.46	2500	1.25	2.50
5	0.58	3000	1.50	3.00
6	0.73	3500	1.75	3.50
7	0.90	4000	2.00	4.00
8	1.05	4500	2.25	4.50
9	1.21	5000	2.50	5.00
10	1.40	5500	2.75	5.50

selected real workflows from four different scientific areas: Montage, CyberShake, SIPHT, and LIGO. Usually, all these workflows use large-scale datasets and memory. Every workflow that has been considered in this work has its unique requirements for data and computing power. The complete representation of these scientific workflows is introduced by [17]. The topological organization of these four scientific workflows is depicted in Figures 1.3, 1.4, 1.6, and 1.7 respectively.

In our experiments, we have focused primarily on three objectives including energy utilization, monetary cost, and resource utilization. These three parameters are studies and presented with varying workflow deadlines. To acknowledge the effectiveness of the proposed ECWS algorithm on reducing energy utilization, cost-saving, and maximizing resource utilization this work performs three popular scheduling algorithms HEFT, EES, and EHEFT, and results compared with our algorithm. A notable list-based scheduling approach HEFT is a heuristic for workflow applications that optimizes the makespan. The HEFT selects tasks based on ranks at each step and assigns a processor for achieving the best EFT with an insertion-based technique. The EES algorithm is a makespan retaining HEFT-based enhanced energy-efficient workflow scheduling approach to minimize energy utilization. The basic thought of the EES algorithm is to exploit the slack associated with the tasks on the non-critical paths and globally allocate them to reduce energy utilization. Enhancing the HEFT approach algorithm is also a HEFT-based workflow task scheduling technique to minimize energy consumption.

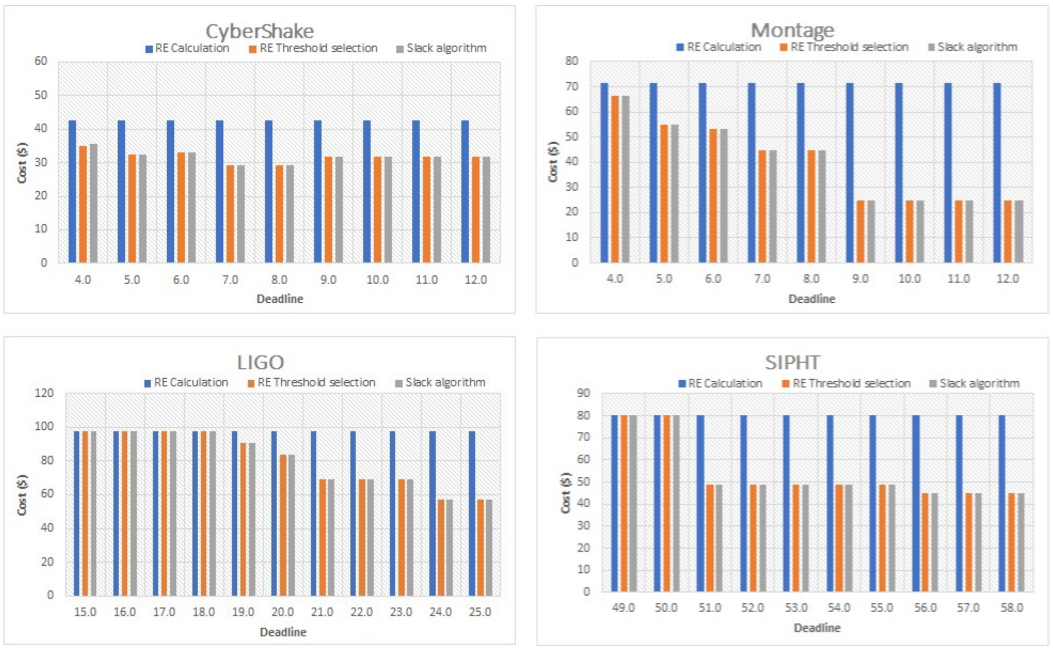

FIGURE 3.2: Self-comparison in cost.

3.5.1 Self-comparison

Our proposed approach has sub-algorithms, we have compared experimental results for energy and cost by taking the RE calculation algorithm as a baseline. It should be emphasized that the RE Threshold selection algorithm contains two sub-algorithms such as the HEFT schedule and RE Calculation algorithms. The experimental results of RE Calculation, RE threshold selection, and slack algorithms for the cost and energy objectives of four scientific workflows are shown in Figure 3.2 and Figure 3.3 respectively.

Compared to the RE calculation algorithm, it is clear that the RE threshold selection algorithms saving more cost and energy. It is because the RE threshold selection algorithm efficient in identifying and switching off the inefficient processors to reduce energy consumption and cost-saving. Moreover, with the increase of the workflow's deadline the energy consumption and cost decrease because based on the deadline the algorithms further removes some inefficient processors. Further, energy reduction with the slack algorithm. It reclaims the slack identified in non-critical tasks by reducing the task running frequency and voltage. But there is no cost reduction with the slack algorithm as it continues with the same leased period of the VM.

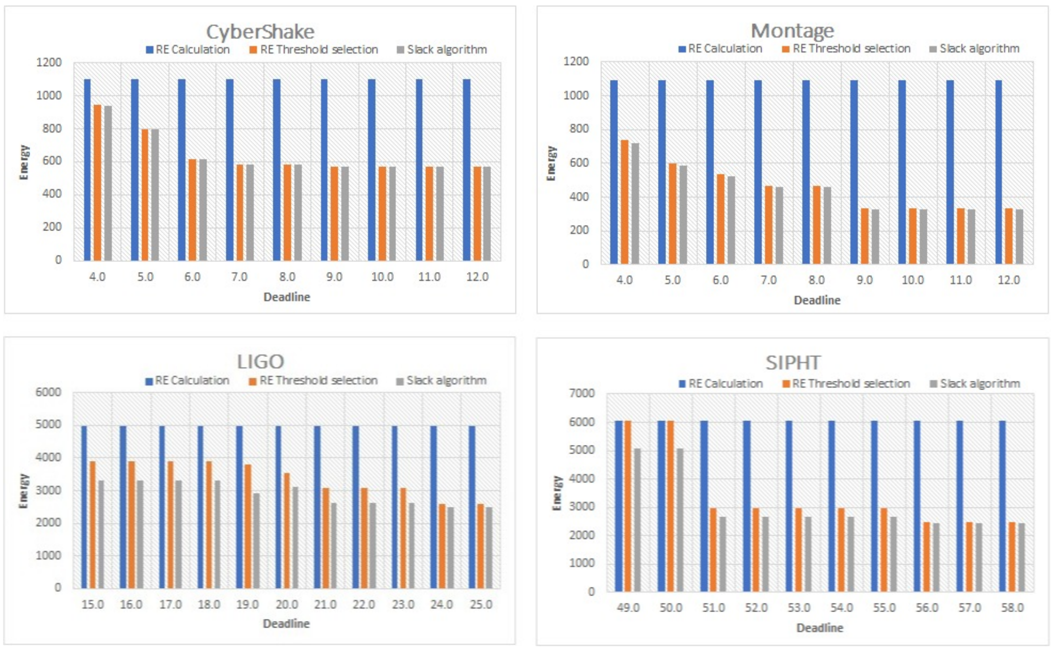

FIGURE 3.3: Self-comparison in energy.

3.5.2 Results comparison with others

The simulation experimental results on the cost of the four different algorithms for four different scientific workflows are shown in Figure 3.4. It is easy to see that the financial costs associated with different algorithms are varied. Among the four algorithms, our proposed ECWS approach has the minimum cost expense. The percentage of cost-saving of the ECWS algorithm over the HEFT, EES, and HEFT on four different workflows is given in Table 3.3. Whereas the performance of the HEFT approach is worst. The EES and EHEFT algorithms have better than the HEFT algorithm. The HEFT and the EES algorithms are independent of the workflow's deadline T_D. Hence, Figure 3.4 has consistent results for the HEFT and the EES algorithms. The reason for this is that the cloud data center has different types of VMs and each type could provide an unlimited number of VM instances to the users. To optimize the overall schedule length of the workflow the scheduling approaches in both HEFT and EES algorithms greedily allocate the computing resources with the highest performance. Consequently, this greedy resource allocation to the workflow tasks regardless of the VM instance type pricing models. Hence it results in excessive monetary costs and is even not influenced by the workflow deadline T_D.

We have evaluated the energy-saving potential of our ECWS algorithm and compared it

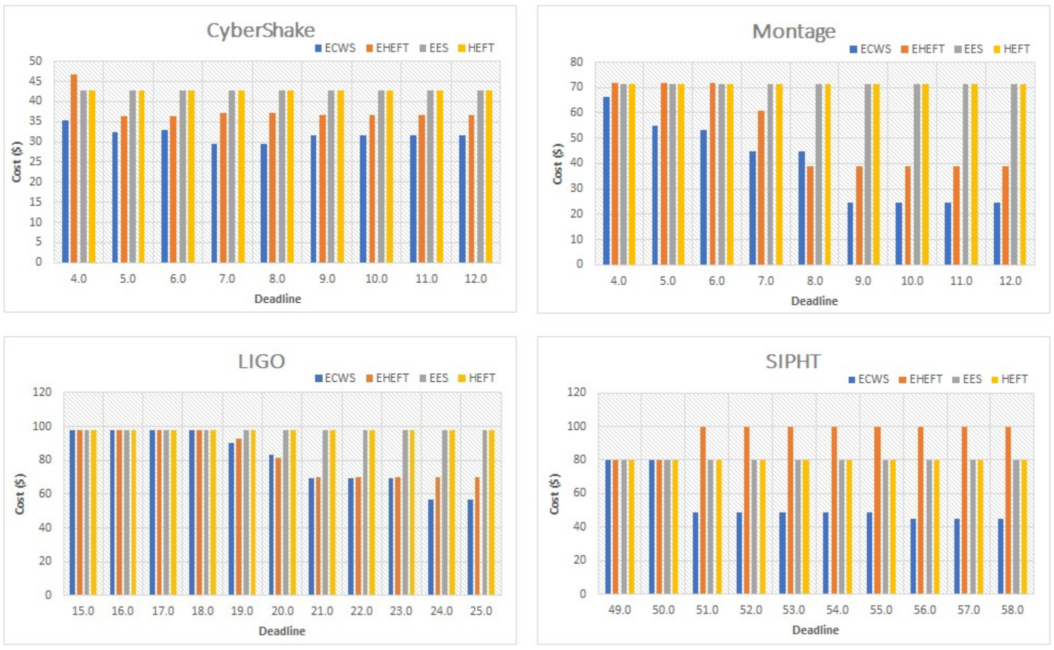

FIGURE 3.4: Comparison with others in cost.

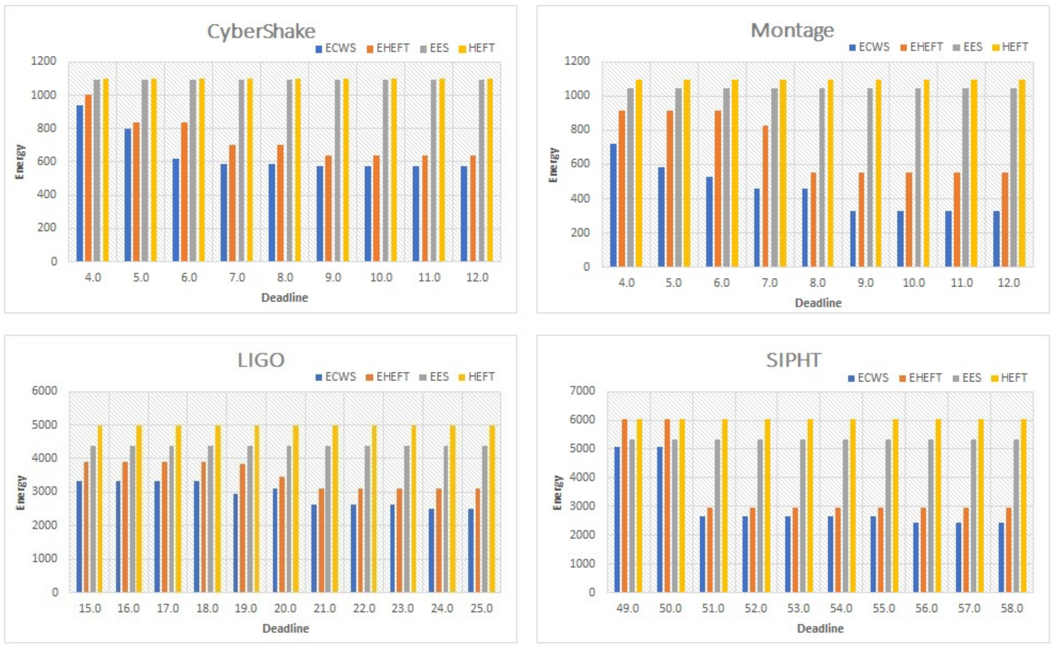

FIGURE 3.5: Comparison with others in energy consumption.

with three other algorithms. The simulation experimental results for the energy optimization of the four algorithms for four different scientific workflows are shown in Figure 3.5. It is prominent that the ECWS algorithm consumes less energy among the four algorithms. Whereas the performance of the HEFT approach is worst, it consumes the highest energy

TABLE 3.3: Cost-efficiency of ECWS over other algorithms

Workflow	EHEFT	EES	HEFT
CyberShake	15.83463	25.33365	25.33365
Montage	22.90583	43.68706	43.6398
LIGO	17.98869	22.43403	22.43403
SIPHT	43.74099	32.84699	32.84699

than the other three. The EES consumes less energy compared to the HEFT algorithm. this is because it reclaims the slack measured in non-critical tasks by applying the DVFS technique. Medium level energy is consumed by the EHEFT algorithms as it finds and switches off power-inefficient processors. Similar to the cost performance HEFT and the EES algorithms are independent of the workflow's deadline T_D. Hence, Figure 3.5 has consistent results for the HEFT and the EES algorithms. The reason behind this the heterogeneous nature of the cloud data center resources (VMs). As discussed in the cost analysis the HEFT and the EES algorithms map the task to the processor (VM instance) having the maximum computing performance without considering the power-efficiency of the VM instance type. This greedy mapping results in the highest power consumption hence energy consumption is not affected by the workflow's deadline T_D. The overall energy saving of the proposed approach over the three considered algorithms (HEFT, EES, and EHEFT) on different workflows are present in Table 3.4. Note that all the values in Table 3.4 are in percentage.

The energy-saving is more with the ECWS algorithm because it efficiently evaluates the ratio of the effectiveness of the processors and selects the best energy giving processors by considering the workflow's deadline along with the slack reclaiming. Moreover, we observed that the ECWS algorithm saves more energy by maximizing resource utilization. The utilization of different components (resources) in a cloud data center impacts power utilization, among them the CPU utilization impacts more on power consumption. The resource utilization performance of the ECWS algorithm is compared with three algorithms as shown in Figure 3.6. We evaluated the percentage of CPU resource utilization of a processor vm_i using the equation (3.14) and the overall percentage of resource utilization for the application using equation (3.15). Similar to energy and cost performance, HEFT and EES algorithm's resource utilization is stable and overlapped in Figure 3.6. The EHEFT performed differently on different workflows as the workflow structures and tasks computing requirements made it switch off more or less number VMs compared with the

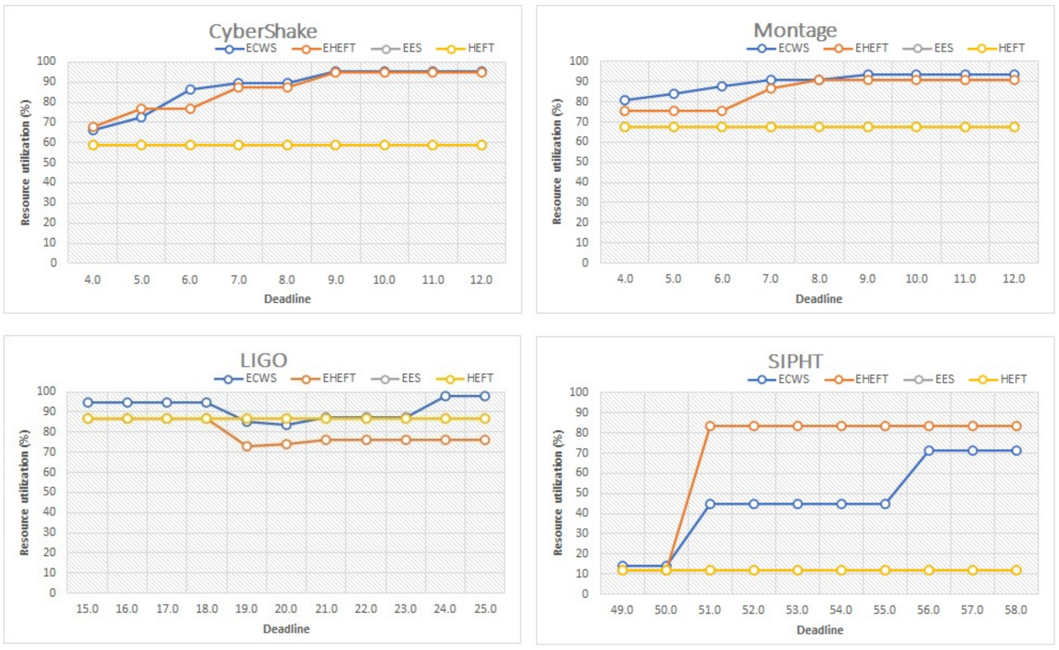

FIGURE 3.6: Comparison experiments in resource utilization.

TABLE 3.4: Energy-efficiency of ECWS over other algorithms

Workflow	EHEFT	EES	HEFT
CyberShake	12.19052	32.72157	32.93524
Montage	35.63741	56.79459	58.58622
LIGO	19.05413	35.83956	43.73791
SIPHT	13.88375	32.71772	40.76495

proposed approach. As an example, the EHEFT algorithm performed well in identifying power inefficient VMs with SIPHT workload and maximized resource utilization on active processors as shown in Figure 3.6. But the ECWS algorithm saved more energy compared to EHEFT on SIPHT workload as shown in Figure 3.5, this is because the ECWS approach effectively reclaimed the slack period associated with non-critical tasks. The average percentage of resource utilization on four workflows over the three other algorithms is depicted in Figure 3.7. In a word, our proposed ECWS algorithm outperformed other approaches in reducing cost, energy-saving, and maximizing resource utilization.

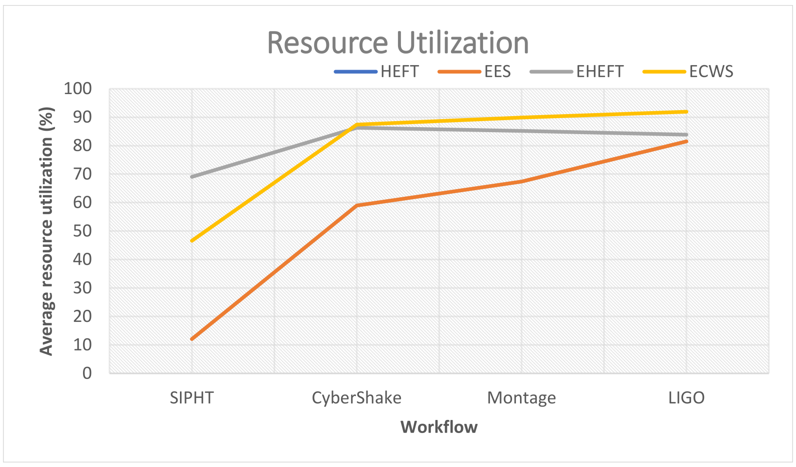

FIGURE 3.7: Average resource utilization.

3.6 Conclusion

In this work, we introduce energy and cost-aware workflow scheduling approach for cloud schedulers. It is efficient in reducing both the monetary cost and the energy consumption and maximizes resource utilization by complying with the user-specified deadlines. Our proposed ECWS algorithm consists of four sub-algorithms. First, the scheduling tasks with the HEFT algorithm for better makespan estimation. Then, RE calculation methods are used to identify the inefficient processors. To switch off the inefficient processors we proposed the RE threshold selection algorithm which effectively switches off inefficient processors considering the application deadline. Lastly, we used a task slacking algorithm to reduce more energy by the DVFS technique. In short, the RE threshold selection algorithm can lower the monetary cost and energy consumption of workflow applications competently. Further, the slacking algorithm and RE threshold selection algorithm can save considerable energy. Experimental results revealed that significant cost and energy saving over EHEFT, EES, and HEFT algorithms. The complete quantitative measurements of the proposed algorithm in cost and energy savings over others with four different workloads are listed in Tables 3.3 and 3.4 respectively.