Energy-aware Workflow Scheduling Using Frequency Scaling

Ilia Pietri, Rizos Sakellariou
School of Computer Science, University of Manchester, UK

***Abstract*—Dynamic Voltage and Frequency Scaling (DVFS) is a power management technique used to decrease the processor frequency and minimize power consumption in modern computing systems. This may lead to higher energy savings for large-scale computational problems, with scientific workflows comprising an important category of applications among these. However, as frequency scaling may result in increased execution time overall, idle time on the processors may also increase, to such a degree that any gains in power are annulled; this depends on the system and workflow characteristics. In this paper, we propose a scheduling algorithm that adopts frequency scaling to reduce overall energy consumption of scientific workflows given an allocation of tasks onto machines and a deadline to complete the execution. Based on the observation that using the lowest possible frequency may not necessarily be energy-efficient, the proposed algorithm works iteratively to scale the frequency further and distribute any slack time, only when overall energy consumption can be decreased. Synthetic data based on parameters of real scientific workflows are used in the evaluation. The results show that the proposed algorithm can achieve energy savings, sometimes at the expense of execution time to reduce the idle time of the processors and decrease overall energy consumption.**

***Keywords*-energy-aware scheduling; frequency scaling; DVFS; DAG scheduling; workflow**

I. Introduction

Energy efficiency is an issue of increasing interest in modern computing systems, such as clouds and clusters, mainly due to the increased operating costs and impact on the environment [1]. Modern processors include features for power management with Dynamic Voltage and Frequency Scaling (DVFS) capabilities being commonly used in order to reduce the processor frequency and decrease power consumption [2]. Processors can operate on a discrete number of frequency states (speeds), each corresponding to a respective voltage level [3]. Lowering the frequency may lead to decreased energy consumption, but this is not always the case, as running an application at lower frequency may lead to increased execution time [1], [4], [5].

Scientific workflows [6] that comprise of tasks with data dependencies between them, typically modelled as a Directed Acyclic Graph (DAG), are an important category of high performance applications under study. The problem of energy-aware scheduling for scientific workflows has received lots of interest with power and performance management algorithms adopting frequency and/or voltage scaling to achieve a power-performance trade-off like in [7], [8], [9], [10]. However, data dependencies between tasks create constraints on scheduling and may limit the energy gain of deploying DVFS techniques. For example, reduction in frequency may reduce power consumption, but may also affect execution time of the workflow. Longer execution time may result in increased energy consumption [1] due to the energy spent while the processors are idle. To reduce overall energy consumption the power consumed by the processors while idle has to be considered [2]. Also, operating the processor to the lowest frequency may not be energy-efficient when the penalty in execution time is higher than the power savings. In such scenarios, running the application using a higher frequency may result in shorter execution time (makespan) consuming less energy [5]. Based on the system and application characteristics it can be determined whether further reduction in frequency brings energy savings. The motivation of this work is based on the observation that using the lowest possible frequency may not always require the least energy to complete the workflow.

In this paper, we address the problem of energy-aware task scheduling of scientific workflows using frequency scaling while meeting a user deadline. The work focuses on heterogeneous machines with different task runtime and frequency capabilities. Given an initial schedule that maps the tasks of the workflow to a number of available machines, the algorithm determines the processor frequency to be used for the execution of each task statically taking into account its slack time (which is the additional time the execution of a task can be stretched without exceeding the deadline), to ensure that the deadline can be met. In contrast to related work, the proposed algorithm works *iteratively* to gradually scale the processor frequency used for each task for as long as overall energy savings are increased. To do so, the next available frequency mode for each processor is used as a lower bound in each iteration to scale the frequency of the tasks.

The most related work can be found in [7], where a scheduling algorithm is proposed to distribute the slack time globally and achieve energy savings by trying to use a uniform frequency for non-critical nearby tasks that run on the same processor. Each task runs using the lowest possible frequency depending on the allocated slack time to ensure that the deadline is met. The main difference of this work is that overall energy consumption is taken into account in the decision making to achieve a trade-off between energy consumption and execution time based on the observation that using the lowest possible frequency may

not be energy-efficient [2], [5]. This is not always true in workflows (DAGs), as in some cases scaling the frequency may lead to performance degradation and increased idle time. In other scenarios, the decrease in power consumption may be lower than the penalty on execution time of the tasks. As a result, energy savings may be limited due to the impact of frequency scaling on execution time [2], [5]. The proposed algorithm can be applied for systems with different power characteristics and determine how frequency scaling can be energy-efficient for the workflow.

The rest of the paper is organized as follows. In Section II related work is presented. In Section III the problem and the assumptions made are described. The proposed scheduling algorithm follows in Section IV, while the experimental evaluation and results are presented in Section V. Section VI concludes the paper.

II. Related Work

Lots of work has been done on task scheduling of workflows to address different user and system constraints including the economic cost of user, execution time and energy consumption [8], [11], [12], [13], [14]. The Heterogeneous Earliest Finish Time (HEFT) algorithm [11] is a well known heuristic among these that deals with workflow scheduling on heterogeneous systems, showing good performance in terms of execution time. Power management techniques and specifically task scheduling using DVFS techniques is the focus in many studies [8], [15], [16], [17], [18]. In [18], an online DVFS mechanism for slack prediction is presented. The algorithm achieves energy savings for HPC applications by finding the slowest frequency so that the critical path is never slowed. In [8], the proposed algorithm adopts DVFS to achieve a trade-off between the quality of the schedule (execution time) and energy consumption. The work focuses on precedence-constrained parallel jobs without deadline constraints. In [17], energy consumption is reduced by extending the execution time of non-critical tasks so that total execution time is not affected. In contrast to these, our algorithm achieves a trade-off between execution time and energy consumption, scaling the processor frequency for each task so that idle time of processors is reduced; this means that we strike a different balance every time, depending on workflow and system characteristics, to minimize overall energy consumption.

A significant amount of work has also been done on slack reclamation to achieve energy savings [3], [7], [9], [10], [19], [20], [21]. Slack reclamation is a technique used to lower the operating frequency and/or voltage of the processor in order to reduce power, sharing the slack time of the tasks effectively. This aims to exploit the idle slots of the processors occurred due to the earlier completion time of the tasks compared with the latest finish time, which is constrained by the deadline and/or data dependencies [8]. In [7], [9], energy-aware algorithms that implement DVFS techniques to reduce energy consumption as uniformly as possible are proposed. Slack reclamation is adopted to scale the processor frequency to the lowest possible frequency for the execution of each task uniformly along with the critical path in the task graph or each processor respectively. The work in [3] is also closely related to slack reclamation approaches, addressing the problem of minimizing energy consumption and investigating the impact of several speed variation models on the complexity of the problem, given a deadline and an initial task scheduling. In [10], a three-phase DVFS algorithm that reduces energy consumption by clustering task slack times using task graph unrolling is proposed. Execution time of the tasks is stretched based on the critical path analysis and the processors power profile. Finally, in [20], a linear combination of the maximum and minimum operating frequencies for the execution of tasks with certain deadlines is proposed. The algorithm provides suitable time portions of each frequency to exploit the slack time of each task, based on the observation that modern processors can operate only on a discrete number of frequencies. As a result, operating a task at a fixed frequency using slack reclamation may not minimize the idle time of tasks. Our paper is closely related to slack reclamation algorithms for workflows with deadlines longer than the execution time of the initial schedule but does not adopt the common assumption of existing work to lower frequency and/or voltage as much as possible. Instead, we lower frequency for individual task execution iteratively, at the same time assessing the overall impact in workflow execution.

III. Problem Description and Assumptions

In our model, the user submits a workflow for execution specifying a deadline for completion of the execution. An initial schedule of tasks on a number of heterogeneous machines is built; this also takes into account data communication between tasks. The initial task schedule is derived assuming the processors operate at the maximum processor frequency. After the assignment of slots to the tasks, the processor frequency for each task can be scaled taking into account its slack time to provide a new schedule that requires less energy consumption and does not exceed the deadline.

Application Model: The work focuses on scientific workflows that can be modelled as Directed Acyclic Graphs (DAGs) with the nodes being the tasks and the edges representing data dependencies between them. The workflow must be executed within a deadline. The runtime of a task t when the processor operates at the maximum frequency f_{max} is given by: $runtime_{t_{f_{max}}} = \frac{size_t}{MIPS}$ and changes to

$$runtime_{t_f} = (\beta_t \cdot (\frac{f_{max}}{f} - 1) + 1) \cdot runtime_{t_{f_{max}}}, \quad (1)$$

when frequency f is assigned [22]. Runtime of the tasks is given in $secs$. The parameter β_t shows the sensitivity

to frequency scaling depending on the CPU-boundedness of the job [5], [22]. The parameter takes values between 0 and 1 and shows the impact of frequency scaling to the execution time of the job. Values close to 1 correspond to CPU-intensive applications, while values close to 0 mean that frequency does not affect execution time of the job [22]. It is assumed that communication time required for the data transfer between the tasks is not sensitive to frequency scaling, being a non-CPU activity [1].

Cloud Resources Model: The work focuses on static scheduling on heterogeneous machines with different capabilities in terms of Millions Instructions Per Second (MIPS) and operating frequency states and which can be provisioned on demand. It is assumed that a task has exclusive control of the processor where it runs. The processor operates at its lowest frequency in idle state when no task has been assigned to it. Transition overhead between different frequencies is not considered, as the frequencies are assigned to the tasks statically and take negligible time compared with the job runtimes [7], [22].

Energy Model: The power consumption, P_f, of each processor operating at frequency f is computed using a cubic model derived in [23] as:

$$P_f = P_{base} + P_{dif} * (\frac{f - f_{base}}{f_{base}})^3, \quad (2)$$

with the power given in $Watts$ and frequency in MHz. In idle state a processor r operates at its lowest frequency consuming idle power, P_{idle_r}. The processors are assumed to be switched on for the whole period of the scheduling (plan). Total energy consumption includes both the dynamic energy required to execute each task t of the workflow w operating at the assigned frequency f_t and the energy spent when the processors are idle (T_{idle_r}). The formula is given in Eq. 3:

$$E = \sum_{\forall t \in w} P_{f_t} \cdot runtime_{t_{f_t}} + \sum_{\forall r \in plan} P_{idle_r} \cdot T_{idle_r} \quad (3)$$

IV. Scheduling Algorithm

The proposed algorithm, Energy-aware Stepwise Frequency Scaling (ESFS), applies frequency scaling to reduce overall energy consumption of scientific workflows based on an allocation of tasks onto machines and a deadline constraint. Before ESFS is invoked, an initial schedule is built, as described below.

Initial Task Scheduling: Firstly, an initial plan to schedule the tasks to the available processors is made using the HEFT algorithm [11]. The slot with the earliest finish time between the available slots of the heterogeneous machines is assigned to each task, taking into account data communication constraints. Communication cost between tasks running on the same processor is considered 0. Each task t is assigned to an initial slot [$startTime_t$, $finishTime_t$]. The duration of the slot, $runtime_t$, depends on the assigned processor r assuming that the processor operates at its maximum frequency.

Algorithm 1 Energy-aware Stepwise Frequency Scaling Algorithm - ESFS.

Require: w: workflow, $curPlan$: HEFT plan, $deadline$: user deadline
Ensure: Apply DVFS to reduce energy consumption
procedure DVFS(w, $curPlan$)
  $curMode = maxMode$, $phase = 1$ and $newPlan = curPlan$
    ▷ $maxMode = max_r fmodes_r - 1$, corresponding to f_{max_r} in each processor r
  while $curMode > 0$ **do** ▷ 0 in the case of f_{min_r} for each r
    $energyCurrent = getCurrentEnergy(curPlan)$;
    $curMode--$ ▷ check for next frequency mode
    $energySortedTasks : \forall t \in w$ ▷ list of tasks to apply DVFS
    while $energySortedTasks$ not empty **do**
      $energySortedTasks = \emptyset$
      $getSlackTimes(newPlan, deadline)$
      for $t \in w$ **do**
        $energyGain_t = getEnergyGain(t, slackTime_t)$
        if $energyGain_t > threshold$ **then**
          ▷ threshold to apply DVFS
          Add t to the $energySortedTasks$ list
        end if
      end for
      Sort the tasks in the list in descending order by energy gain
      Remove the first task, t, from the list
      Set $f = minf_i \in [\max(f_{curMode_r}, f_{min_r}), f_t)$ with $runtime_{t_{f_i}} < slotGap$ and $slotGap = runtime_{t_{f_t}} + slackTime_t$
      Update task runtime using Eq. 1 and finish time for $f_t = f$
      $newPlan$: Update the slots of the tasks in the new plan
    end while
    $energyNew = getCurrentEnergy(newPlan)$;
    if $energyNew >= energyCurrent$ && $phase == 1$ **then**
      ▷ Reduction in frequency doesn't bring energy gain
      Reject $newPlan$ and return to the previous mode: $curMode++$
      Set $deadline = makespan_{curPlan}$ and $phase = 2$
        ▷ the loop continues with phase 2
    else
      Accept plan ($curPlan = newPlan$)
    end if
  end while
end procedure

Energy-aware Stepwise Frequency Scaling Algorithm - ESFS: After the initial assignment of slots (initial plan), the ESFS algorithm (Alg. 1) works iteratively to scale the assigned frequencies gradually and distribute the slack time to the tasks in an energy-efficient way. To do so, a counter, $curMode$, is used to set a lower bound in each iteration for the operating frequency to be assigned for the execution of each task. This also allows to deal with the heterogeneity of machines, as processors may operate on different frequencies and number of frequency states (frequencies), $fmodes$. Initially, all the processors run at their maximum frequency ($curMode = maxMode$). In each iteration the next available mode ($curMode--$), which corresponds to the next lower frequency of each processor, is used as a bound. This is done until the frequency mode is equal to 0, which corresponds to the minimum available frequency for all the processors. When a processor does not include lower

frequency states, the minimum processor frequency is used as a bound for the tasks assigned to it in order to deal with the heterogeneity of the machines.

When the deadline is longer than the makespan of the initial plan, frequency scaling may result in longer execution time which may not be energy-efficient. To avoid this, the algorithm works in two phases. In phase 1 frequency scaling is applied as long as overall energy consumption decreases in order to produce an energy-aware plan that meets the user deadline. Then, the algorithm continues with phase 2 to reduce the frequency further when higher energy savings can be achieved without affecting the workflow execution time. To do so, the deadline in phase 2 is set equal to the makespan of the current accepted plan (line 25 in Alg. 1).

The procedure that takes place in each iteration is the following: The energy consumption of the current plan, $energyCurrent$, is computed using the energy model described in Eq. 3 for the whole period (makespan) of the plan, $curPlan$ (line 4). The next frequency mode is used to check if frequency scaling can be used to reduce overall energy consumption (line 5). To do so, the slack time of each task, $slackTime_t$, which indicates the maximum value that can be added to the task execution time without exceeding the deadline, is computed recursively in an upwards function (line 9) like in [24]:

$$slackTime_t = \min_{s \in suc_t} (spareTime_{t \to s} + slackTime_s), \tag{4}$$

where $spareTime_{t \to s} = startTime_s - finishTime_t - comCost_{t \to s}$ shows the maximum delay in the execution of the task that will not affect the execution of its successors (in the DAG and the processor). The communication cost, $comCost_{t \to s}$, is considered to be 0 when the tasks t and s are assigned to the same processor. The slack time for the exit node is set equal to the difference between the deadline and the finish time of the task: $slackTime_{t_{exit}} = deadline - finishTime_{t_{exit}}$.

Taking into account the slack time of each task, its energy gain (line 11) from the transition to a lower frequency (if any) can be computed in the following way: Firstly, the time the task can run without exceeding the deadline, $slotGap$, is computed as the sum of the current execution time and slack time of the task in order to find the lowest frequency f in the iteration that can be applied for the task so that the deadline is not exceeded. The idea is that when operating the processor in higher frequency f_{cur} with the execution time of the task given by $runtime_{f_{cur}}$, the processor remains idle for the slack time of the task. This idle periods of the processor can be exploited to stretch the execution of the task if needed and lower the assigned frequency to achieve energy gain. When the processor operates at the lower frequency f, the runtime of the task is equal to $runtime_f$. The transition to a lower frequency may lead to an energy gain from the difference between the energy consumption in the two

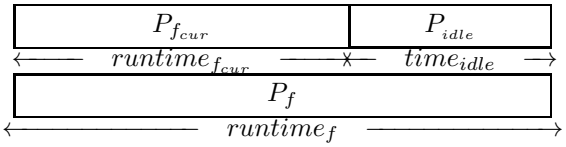


Figure 1: Task execution in different frequencies.

cases. Fig. 1 presents the two different scenarios of running the task using the current assigned frequency f_{cur} and the lower frequency f respectively when there is slack time. The energy gain for a task when scaling the processor frequency f_{cur} to a lower frequency f is computed as

$$energyGain = \frac{(E_{f_{cur}} - E_f)}{E_{f_{cur}}}, \tag{5}$$

where $E_{f_{cur}} = P_{f_{cur}} \cdot runtime_{f_{cur}} + P_{idle} \cdot (runtime_f - runtime_{f_{cur}})$ is the energy spent when the task runs at processor frequency f_{cur} with the processor operating at idle state for the remaining time in the observed period and $E_f = P_f \cdot runtime_f$ is the energy required when the task runs using the lower frequency f.

The tasks with energy gain, $energyGain_t$, larger than a given threshold are ordered in descending order and the most energy promising task is removed from the list to apply frequency scaling (lines 17, 18). The slot of the task is then updated (line 19). As start time of the successors of the tasks may be affected (in case the execution of the task required more than its available spare time), the slots of the remaining tasks are updated and the new plan, $newPlan$, is provided (line 20). Lines 9-20 are repeated to update the list with the tasks for frequency scaling until there is no other task in the list for the current frequency bound.

In the end of each iteration the energy consumption of the plan is computed (line 22) and used to determine if overall energy consumption decreases with the transition to a lower frequency mode. If energy savings are achieved the current plan is accepted (line 27) and the procedure continues to the next iteration. Otherwise, when the transition that may result in longer execution time (makespan) does not bring energy savings, the plan is not accepted and the algorithm continues to phase 2 (line 23). In this case, the deadline is set equal to the makespan of the current plan (line 25) and the loop continues with the current mode. The procedure described earlier (for phase 1) is followed until the lowest mode (equal to 0) is reached so that higher energy savings can be achieved.

A. Example

An example with three different scheduling approaches, using the DAG shown in Fig. 2, is described to demonstrate the motivation of this work and explain how the algorithm works. Each node is annotated with the id of the task and its

processor $r0$	processor $r1$
0 (0-1300, 2600)	
2 (1300-2600, 2600)	1 (1300-2200, 2600)
4 (2600-3800, 2600)	3 (2200-2900, 2600)
5 (3800-4800, 2600)	6 (2900-4000, 2600)
7 (4800-6100, 2600)	
makespan: 6100 secs energy: 0.65 kWhs	

(a) Initial schedule.

processor $r0$	processor $r1$
0 (0-1877.8, 1800)	
2 (1877.8-3755.6, 1800)	1 (1877.8-3177.8, 1800)
4 (3755.6-5488.9, 1800)	3 (3177.8-4188.9, 1800)
5 (5488.9-6933.3, 1800)	6 (4188.9-5777.8, 1800)
7 (6933.3-8811.1, 1800)	
makespan:8811.1 secs energy:0.74 kWhs	

(b) Processor operating at lowest frequency.

processor $r0$	processor $r1$
0 (0-1408.33, 2400)	
2 (1408.3-2816.7, 2400)	1 (1408.3-2708.3, 1800)
4 (2816.7-4116.7, 2400)	3 (2708.3-3719.4, 1800)
5 (4116.7-5200, 2400)	6 (3719.4-5149.4, 2000)
7 (5200-6608.3, 2400)	
makespan:6608.3 secs energy:0.63 kWhs	

(c) ESFS schedule.

Figure 3: Different scheduling approaches.

size in Millions Instructions: task id (task size). The three scheduling scenarios are shown in Fig. 3a, 3b and 3c and correspond to: (a) the initial plan of HEFT; (b) the modified schedule when the processors operate at the minimum frequency, and; (c) the schedule developed using the ESFS algorithm respectively. In each case, the assignment of tasks in Fig. 3 shows: task id (start time - finish time, operating frequency). For simplicity, two homogeneous processors of 1 MIPS and five operating modes in steps of 200 MHz in the range of 1800-2600 MHz are used. The communication time is considered to be 0 and β is equal to 1 for all the tasks. The parameters of the power model used to compute the maximum power consumption at each frequency state in Eq. 2 are obtained by [23], while P_{idle} is set equal to 60% of the power $P_{f_{max}}$ consumed at the maximum frequency of each processor. The deadline is equal to 1.5 times the initial makespan.

Initially the schedule using HEFT is provided to map the tasks to the processors, assuming that each processor operates at its maximum frequency; this assignment is shown in Fig. 3a. Based on the initial plan, a schedule with the processors operating at the lowest frequency is made and presented in Fig. 3b. It can be noted that operating the processors at the minimum available frequency $f = 1800$ MHz to execute all the tasks leads to a longer schedule that requires more energy compared to the initial plan. The schedule using the proposed algorithm, ESFS, is presented in Fig. 3c. The algorithm scales the frequency of the tasks so that overall energy decreases. More specifically, during phase 1 the frequency assigned for the execution of each task is scaled to the next lower frequency to exploit slack time. The plan with the processors operating at 2400 MHz is accepted as overall energy consumption decreases. In the next iteration, the next frequency mode corresponding to 2200 MHz is used as a lower bound and lines 8-20 are repeated to compute the energy gain of each task in the list and reduce the assigned frequency to exploit the slack time. However, the transition to a lower frequency (2200 MHz) leads to increased energy consumption due to the increase in the workflow execution time. As reducing the frequency further does not lead to higher energy savings, the new plan is rejected. The deadline is set equal to the current makespan and the algorithm continues with phase 2. During this phase, the frequency assigned for the execution of tasks 1, 3 and 6 is further reduced, as by doing so there is an overall energy saving without increasing the makespan further. Overall, the schedule from the deployment of the ESFS algorithm, shown in Fig. 3c, results in smaller energy consumption at the cost of higher execution time, when compared with the initial schedule in Fig. 3a.

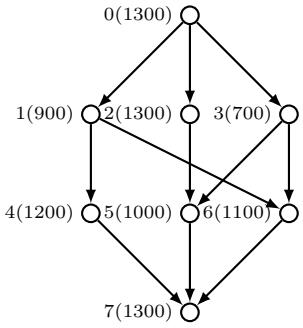


Figure 2: The DAG used in the example.

V. Experimental Evaluation and Results

In this section the proposed scheduling algorithm is compared and evaluated using the Enhanced Energy-Efficient Scheduling (EES) algorithm developed in [7], [25]. The application and system characteristics used in the evaluation are described next.

A. Methodology

The simulator in [26] was used in order to implement and evaluate the proposed algorithm. Two types of heterogeneous machines, which we refer to here as Slow and Fast, are used in the evaluation assuming 1.00 and 1.20 MIPS respectively in the calculation of the execution time of the tasks when the processor operates at its maximum frequency. The frequencies (in MHz) that correspond to each frequency

fmode	0	1	2	3	4
Slow		1800	2000	2200	2400
Fast	1800	2000	2200	2400	2600

Table I: Frequency capabilities of each processor type.

mode are shown in Table I. The power model described in Eq. 2 is used with $P_{base} = 152.00W$, $P_{dif} = 15.39$ and $f_{base} = 1000MHz$, like in [23]. Idle power, P_{idle}, is assumed to be about 60% of $P_{f_{max}}$ based on [5], [23], [27] and set equal to 117 and 129 W for the Slow and Fast type respectively. A threshold of 0.01% was used in line 12 of Alg. 1 so that frequency scaling is applied only to tasks for which scaling brings an energy gain for the task. In general, different values of the threshold may be used to achieve a trade-off between execution time and energy savings. Finally, for the calculation of the communication costs a network of 1 Gbps is assumed.

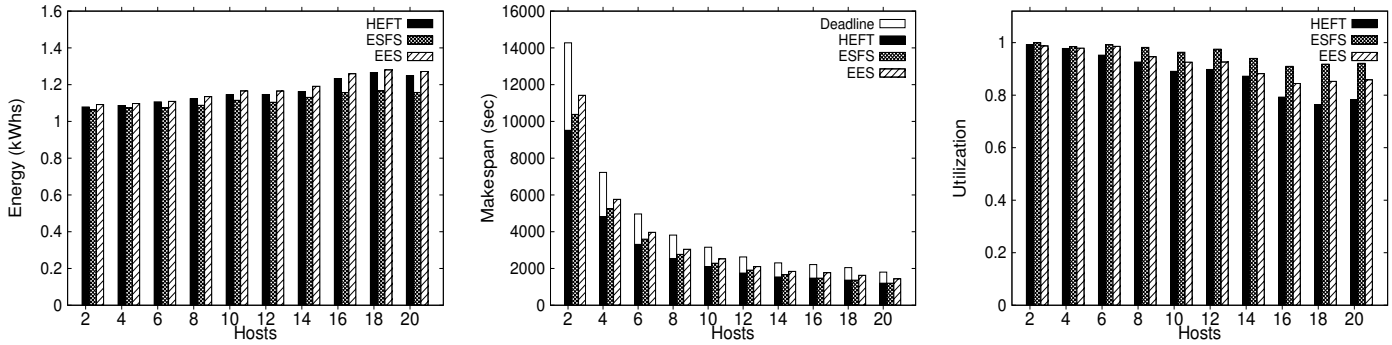
Data from three real scientific applications, namely LIGO [28], SIPHT [28], [29] and Montage [29] were used in the experiments. LIGO is a data-intensive scientific workflow for the detection of gravitational waves in the universe, processing a large amount of data. SIPHT is a computationally-intensive workflow searching for sRNA encoding genes for bacterial replicons. Finally, Montage that generates image mosaics of the sky can be characterized as I/O intensive. Synthetic data [30] based on information from real scenarios is used to create a workflow of 100 tasks for each scientific application. For simplicity, it is assumed that β is equal to 1 for all of the jobs (tasks), a conservative approach when the parameter is not known before scheduling [22]. Two values for the user deadline are used: 1.2 and 1.5 times the makespan obtained from HEFT. Also, a different number of available machines (2 to 20) is used in each experiment to schedule the tasks to the processors and evaluate the performance of the algorithm under different resource utilization scenarios. In each scenario the same number of machines is used for each processor type, Slow and Fast, with the total number of machines used for the workflow scheduling shown on the x-axis in the graphs. For example, in the case of 20 available machines, 10 machines of each type, Slow and Fast, are used. The y-axis includes the results for the total energy consumption given in kWhs, the achieved makespan in secs and utilization, computed as the percentage of the total time required for the execution of the tasks to the total time the processors are switched on. Each processor is switched on for a period equal to the makespan.

B. Results

The performance of our proposed algorithm is compared with the performance of the EES algorithm [7] in terms of overall energy consumption, achieved makespan and utilization. The initial schedule obtained from HEFT is used as a baseline, with each processor operating at its maximum frequency while active and transiting to idle state when no task has been assigned to it. The results are shown in Fig. 4-6 and 7-9 for LIGO, SIPHT and Montage when the deadline is set equal to 1.2 and 1.5 times the makespan of the initial plan respectively. The plots showing makespan also show the deadline on the top of the HEFT bar. A different number of machines (2 to 20 hosts) was also used. In the case of LIGO and Montage, execution time decreases when using additional hosts, but this results in lower utilization. In the case of SIPHT, utilization decreases at a higher rate, as using more than 6 hosts does not improve its execution time.

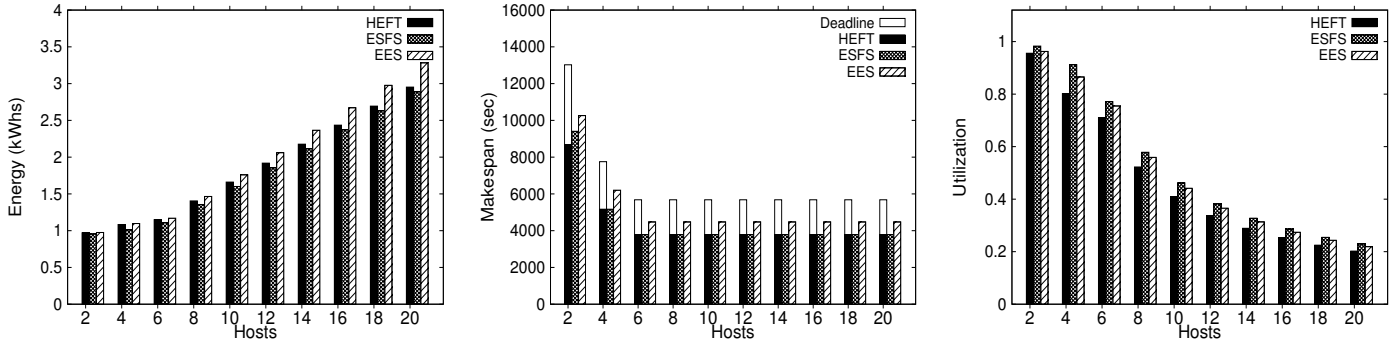
When the deadline constraint is strict (1.2 times), as shown in Fig. 4-6, the proposed algorithm, ESFS, results in smaller energy consumption when compared with EES. This is because the algorithm takes overall energy consumption into account to scale the frequency of the tasks and exploits the slack time to stretch the execution time of the workflow only when higher energy savings are achieved. When frequency reduction does not result in higher energy savings the algorithm continues to phase 2 scaling the frequency of the tasks so that makespan is not increased. Reducing the operating frequency for a task may create idle slots in other processors due to data dependencies between the tasks, which may lead to lower utilization. Additionally, as processors operate at a discrete and/or limited number of available frequency states, slack time may not be fully utilized and idle time may be substantial. This means that operating the processor at a lower frequency may be energy consuming, as idle power is still significant in current systems [27]. The achieved makespan and utilization explain better the results, with workflow execution time being smaller in the case of ESFS. In scenarios with lower utilization (using a large number of hosts), ESFS achieves higher energy savings compared with EES. The impact of frequency scaling on execution time may lead to lower energy savings, with the cost in idle energy exceeding the energy gain from the tasks. In this case, the idle time of the processors can be exploited to scale the frequency without affecting the deadline.

For the long deadline (1.5 times, Fig. 7-9), the difference in energy consumption between the two algorithms is larger. EES takes into account the slack time to distribute it effectively to the nearby tasks resulting in increased makespan, which may also lead to increased idle time. Depending on the structure of the workflow and the nature of the data dependencies between the tasks, idle time result in lower resource utilization. ESFS keeps the makespan smaller considering the impact of workflow execution time on the overall energy consumption. The algorithm achieves increased utilization, keeping the plan that leads to energy savings. Overall, energy savings may differ for different workflows and number of available hosts. The workflow structure and characteristics affect the utilization of the running hosts. As a result, the cost on workflow execution time required to achieve higher utilization and/or energy savings may differ. ESFS controls performance degradation in scenarios where energy consumption does not decrease with the reduction in frequency. As a result, ESFS is not so sensitive to deadlines and does not utilize all the slack to a deadline as this does not necessarily lead to the smallest



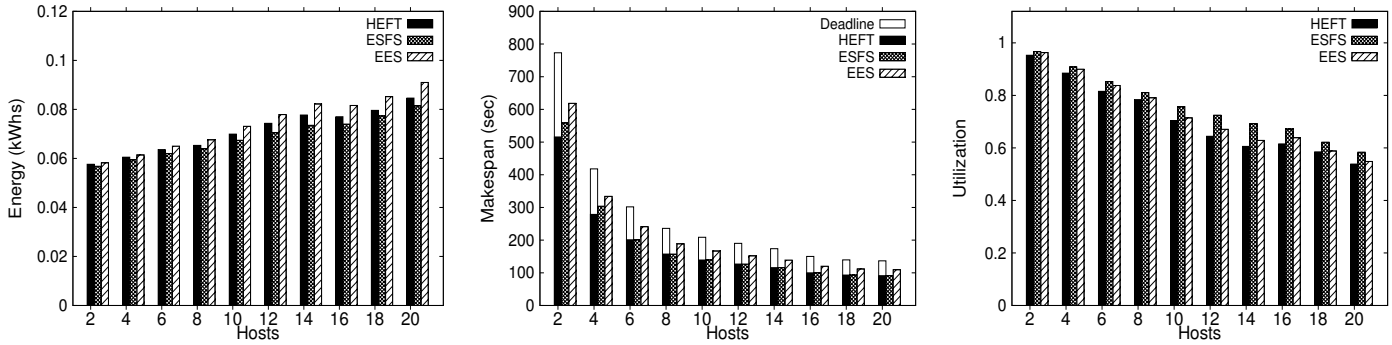
(a) Energy (b) Makespan (c) Utilization

Figure 4: LIGO workflow with 100 tasks and deadline equal to $1.20 \cdot M_{HEFT}$.



(a) Energy (b) Makespan (c) Utilization

Figure 5: SIPHT workflow with 100 tasks and deadline equal to $1.20 \cdot M_{HEFT}$.



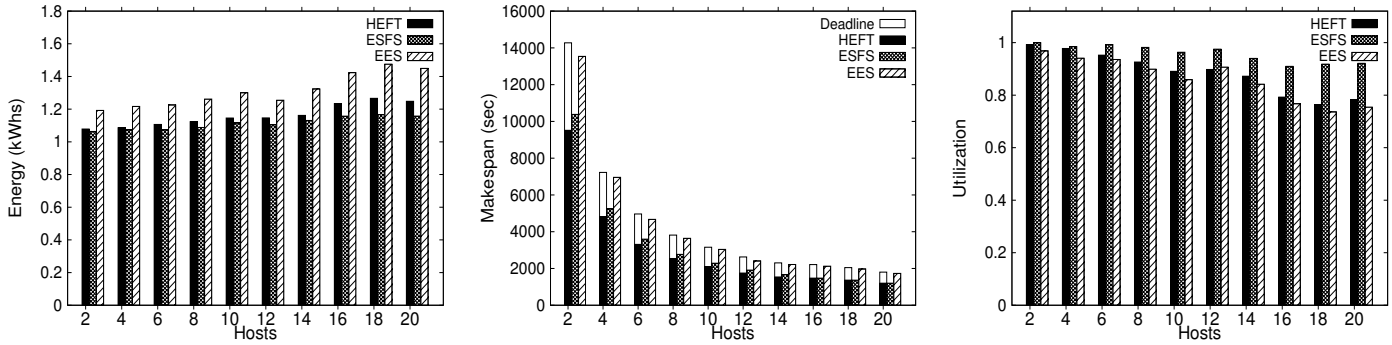
(a) Energy (b) Makespan (c) Utilization

Figure 6: Montage workflow with 100 tasks and deadline equal to $1.20 \cdot M_{HEFT}$.

energy consumption.

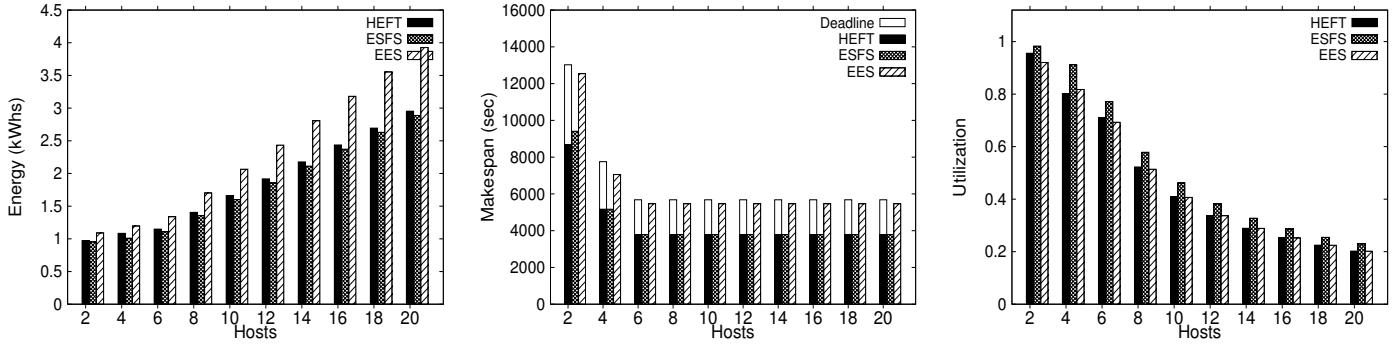
C. Summary of Observations

Scaling each task's operating frequency does not necessarily bring energy savings. This is mainly due to the energy consumed when a processor is in idle state due to data dependencies and the limited number of frequency states of the processors creating constraints on slack time utilization. Workflow and processor characteristics also need to be considered to determine whether further reductions in frequency bring overall energy savings. In some scenarios reducing the frequency may lead to increased execution time and lower utilization, which leads to an overall increased energy consumption. In this case using higher frequencies to run the tasks and set the processors to idle state for the remaining time may be required. In other scenarios, frequency scaling may lead to lower energy consumption and increased utilization of resources. Thus, determining a frequency for each task largely depends on the workflow structure and its characteristics as well as the system characteristics; the



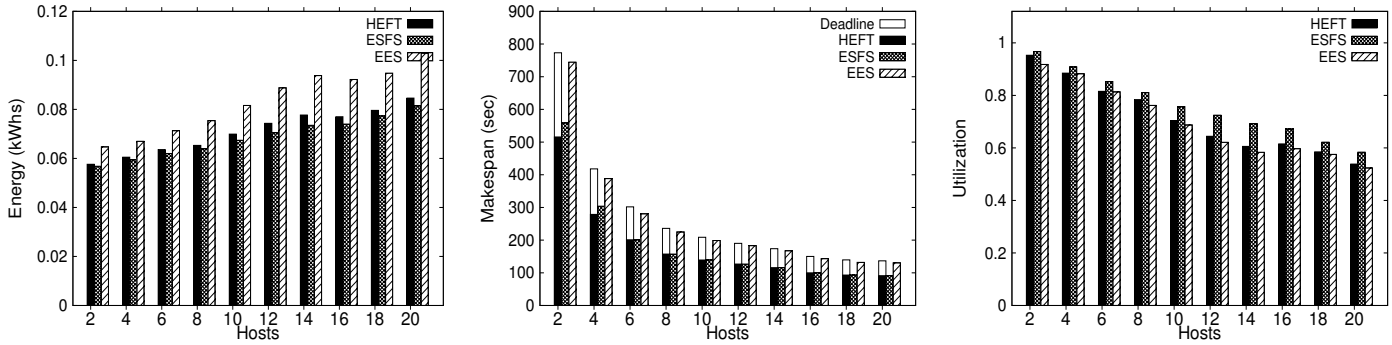
(a) Energy (b) Makespan (c) Utilization

Figure 7: LIGO workflow with 100 tasks and deadline equal to $1.50 \cdot M_{HEFT}$.



(a) Energy (b) Makespan (c) Utilization

Figure 8: SIPHT workflow with 100 tasks and deadline equal to $1.50 \cdot M_{HEFT}$.



(a) Energy (b) Makespan (c) Utilization

Figure 9: Montage workflow with 100 tasks and deadline equal to $1.50 \cdot M_{HEFT}$.

advantage of our proposed algorithm is that it is trying to exploit all of these.

VI. Conclusion

This paper considered the problem of energy-aware task scheduling for scientific workflows under a given deadline. An algorithm using frequency scaling was proposed to determine the operating frequency for each task of the workflow and reduce overall energy consumption. The proposed algorithm, ESFS, works iteratively to distribute the slack time between the tasks and scale the frequency gradually, taking into account how overall energy consumption changes with the reduction in frequency. The performance of the algorithm was evaluated using simulation. The results show that the algorithm can strike a good balance between energy consumption and execution time. Depending on the workflow and system characteristics, different operating frequencies may be required for the workflow execution in order to increase energy efficiency. The current analysis does not consider scenarios where job sensitivity to frequency

scaling, which shows the impact of frequency scaling to the execution time of each job, varies. Future work can investigate the effect of job sensitivity on the performance of the algorithm as well as the impact of different network characteristics and communication costs.

REFERENCES

[1] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "Understanding the future of energy-performance trade-off via DVFS in HPC environments," *JPDC*, vol. 72, no. 4, pp. 579–590, 2012.

[2] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar, "Critical power slope: understanding the runtime effects of frequency scaling," in *Proceedings of the 16th ICS*. ACM, 2002, pp. 35–44.

[3] G. Aupy, A. Benoit, F. Dufossé, and Y. Robert, "Reclaiming the energy of a schedule: models and algorithms," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 11, pp. 1505–1523, 2013.

[4] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the International conference on Power aware computing and systems*. USENIX Association, 2010, pp. 1–8.

[5] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE TPDS*, vol. 18, no. 6, pp. 835–848, 2007.

[6] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *FGCS*, vol. 25, no. 5, pp. 528–540, 2009.

[7] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, and X. Huang, "Enhanced energy-efficient scheduling for parallel applications in cloud," in *Proceedings of the 12th IEEE/ACM CCGrid*. IEEE, 2012, pp. 781–786.

[8] Y. C. Lee and A. Y. Zomaya, "Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling," in *Proceedings of the 9th IEEE/ACM CCGrid*. IEEE, 2009, pp. 92–99.

[9] H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi, "Emprical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster," in *Proceedings of the IEEE CLUSTER*. IEEE, 2006, pp. 1–10.

[10] M. Qiu, Z. Ming, J. Li, S. Liu, B. Wang, and Z. Lu, "Three-phase time-aware energy minimization with DVFS and unrolling for chip multiprocessors," *JSA*, vol. 58, no. 10, pp. 439–445, 2012.

[11] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE TPDS*, vol. 13, no. 3, pp. 260–274, 2002.

[12] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in IAAS clouds," in *Proceedings of the SC*. IEEE, 2012.

[13] T. Thanavanich and P. Uthayopas, "Efficient energy aware task scheduling for parallel workflow tasks on hybrids cloud environment," in *Proceedings of the ICSEC*. IEEE, 2013, pp. 37–42.

[14] I. Pietri, M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, and R. Sakellariou, "Energy-constrained provisioning for scientific workflow ensembles," in *Proceedings of the 3rd CGC*. IEEE, 2013, pp. 34–41.

[15] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE TPDS*, vol. 22, no. 8, pp. 1374–1381, 2011.

[16] R. Mishra, N. Rastogi, D. Zhu, D. Mossé, and R. Melhem, "Energy aware scheduling for distributed real-time systems," in *Proceedings of the IPDPS*. IEEE, 2003.

[17] L. Wang, G. Von Laszewski, J. Dayal, and F. Wang, "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS," in *Proceedings of the 10th IEEE/ACM CCGrid*. IEEE, 2010, pp. 368–377.

[18] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: making DVS practical for complex HPC applications," in *Proceedings of the 23rd ICS*. ACM, 2009, pp. 460–469.

[19] D. Zhu, R. Melhem, and B. R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE TPDS*, vol. 14, no. 7, pp. 686–700, 2003.

[20] N. B. Rizvandi, J. Taheri, A. Y. Zomaya, and Y. C. Lee, "Linear combinations of DVFS-enabled processor frequencies to modify the energy-aware scheduling algorithms," in *Proceedings of the 10th IEEE/ACM CCGrid*. IEEE, 2010, pp. 388–397.

[21] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya, "Some observations on optimal frequency selection in DVFS-based energy consumption minimization," *JPDC*, vol. 71, no. 8, pp. 1154–1164, 2011.

[22] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "Optimizing job performance under a given power constraint in HPC centers," in *Proceedings of the IGCC*. IEEE, 2010, pp. 257–267.

[23] J.-M. Pierson and H. Casanova, "On the utility of DVFS for power-aware job placement in clusters," in *Proceedings of Euro-Par*. Springer, 2011, pp. 255–266.

[24] R. Sakellariou and H. Zhao, "A low-cost rescheduling policy for efficient mapping of workflows on grid systems," *Scientific Programming*, vol. 12, no. 4, pp. 253–262, 2004.

[25] S. Su, Q. Huang, J. Li, X. Cheng, P. Xu, and K. Shuang, "Enhanced energy-efficient scheduling for parallel tasks using partial optimal slacking," *The Computer Journal*, 2014.

[26] Cloud Workflow Simulator, Available:https://github.com/malawski/cloudworkflowsimulator.

[27] A. Kansal and F. Zhao, "Fine-grained energy profiling for power-aware application design," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 2, pp. 26–31, 2008.

[28] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Proceedings of the 3rd WORKS*, 2008, pp. 1–10.

[29] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *FGCS*, vol. 29, no. 3, pp. 682–692, 2013.

[30] Workflow Generator, Available:https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.