SPECIAL ISSUE PAPER

An adaptive deadline constrained energy-efficient scheduling heuristic for workflows in clouds

Wei Zheng,*† and Shouhui Huang

1Department of Computer Science, School of Information Science and Technology, Xiamen University, China

SUMMARY

There is an increasing interest for cloud services to be provided in a more energy-efficient way. The growing deployment of large-scale, complex workflow applications onto cloud computing hosts are facing with crucial challenges in reducing the energy consumption without violating certain quality of service. Dynamic voltage and frequency scaling (DVFS) is a power management technique commonly used to lower the processor frequency and decrease the energy consumption in modern computing systems. However, as lowering processor frequency may result in increased idle time on processors, which may in turn lead to increased overall energy consumption, scaling the processor frequency as low as possible may not always be energy-efficient. In this paper, we consider cloud hosts with the DVFS technique and focus on the problem of scaling frequency to reduce overall energy consumption of a workflow given an allocation of tasks to hosts and a deadline to complete the execution. We propose a novel scheduling heuristic, which takes the system and application characteristics and the overall energy consumption into account when making frequency scaling decision. The proposed heuristic is evaluated using simulation with four different real-world applications. The observed results indicate that our heuristic can achieve significant energy saving and outperform the existing approaches. Copyright © 2015 John Wiley & Sons, Ltd.

Received 26 May 2015; Accepted 16 June 2015

KEY WORDS: workflow scheduling; DVFS; cloud computing

1. INTRODUCTION

The past decades have witnessed large-scale computing platforms (such as grids and clouds) become increasingly popular for delivering various IT services to customers worldwide. This fulfills the explosively growing requirement on large-scale computation and storage arising from various scientific and/or industrial fields (e.g., [1, 2]). However, data centers hosting cloud services consume tremendous amounts of energy. For instance, Google, as a famous cloud service provider, published their energy consumption in 2011 to have been 2,675,898 MWh [3], which was roughly as much electricity globally as 220,000 people, based on electricity use per capita in the USA. [4]. This, as a result, contributes to high operational costs and carbon footprints to the environment [5]. Indeed, there is an urgent need of reducing the energy consumption while still meeting the cloud service requirements from customers.

With rapid deployment of cloud infrastructures, there is a growing interest in executing large-scale, complex applications such as workflow [6] in clouds. A workflow normally consists of a number of inter-dependent tasks (i.e., computing modules), which are often represented by a directed acyclic graph (DAG). Typically, a user may require his/her workflow application to complete...
within a certain deadline, which is commonly recognized as an important cloud service requirement. In this paper, we focus on workflow (DAG) scheduling in order for reducing energy consumption while still meeting deadline constraint.

Techniques like dynamic voltage and frequency scaling (DVFS) have enabled most of cloud hosts, which could be real or virtual and operate on different voltage/frequency states with different capacities. When a schedule of workflow, which is generated based on the full capacity of cloud hosts, results in an overall completion time earlier than the deadline, we may possibly alter the initial schedule to let workflow tasks run on lower capacities of cloud hosts, so as to save energy consumption while still meeting the deadline constraint. In this case, the problem of workflow scheduling boils down to the DAG-scheduling problem in order to minimize energy consumption with a certain constraint of deadline, based on DVFS-enabled hosts.

In general, the DAG-scheduling problem, including the one we study in this paper, is NP-Hard [7]. This means there may not exist an algorithm which can find an optimal solution within polynomial time. As a result, over recent years, research efforts have been tried on developing heuristics to strike a good balance between energy saving and complexity [8] while keeping the deadline constraint satisfied. Most of these heuristics consider the aforementioned DVFS technique, which is commonly included in modern processors in order to reduce the processor frequency and decrease energy consumption. With DVFS, processor can operate on a discrete number of frequency states (speeds), each corresponding to a respective voltage level. It is expected that lowering the frequency may lead to decreased energy consumption.

However, running an application at lower frequency may lead to increased execution time, especially for scientific workflows with data dependencies between workflow tasks. These dependencies create constraints on the sequence of task execution in a schedule. It is not unusual that a processor may be left idle because a task allocated to the processor cannot start execution till all data dependencies of the task are resolved. In such a case, longer execution time may result in increased energy consumption due to the extra energy consumed while the processors are idle [5]. This indicates the following: (1) in order to reduce overall energy consumption of a workflow, the energy consumed by the processors while idle has to be considered; (2) operating the processor to a lower frequency may not be energy-efficient when the penalty in execution time is higher than the energy savings [9]. Majority of the works till today consider ‘time’ as the only factor when making scheduling decision for energy saving. This results in overlooking of the fact that sometimes decreasing the frequency of workflow tasks may render that the overall energy consumption increases. In such a scenario, these heuristics are unlikely to have a good performance. This motivates our work to take the impact of lowering frequency on overall energy consumption into account when scheduling.

This paper is an extended version of a paper that first appeared in [10], where a novel DAG-scheduling algorithm for energy-efficient named group then individual (GTI) is proposed. In this paper, we further investigate how the system and application characteristics affect the scheduling decision and extend the GTI heuristic to the adaptive group then individual (AGTI) heuristic. Moreover, we add a newly published related work Energy-aware Stepwise Frequency Scaling Algorithm (ESFS) [11] into our evaluation. Results from simulation experiments show that in most cases, our scheduling approach outperforms its competitors on energy saving, while still meeting deadline constraint.

The rest of the paper is organized as follows. Related research efforts are summarized in Section 2. Section 3 describes the modeling of the target system, application and energy calculation. Section 4 presents the details of our approaches with a simple motivational example. Section 5 discusses results obtained from simulation experiments. Section 6 concludes this paper and provides a view of our future work.

2. RELATED WORK

Directed acyclic graph scheduling is a widely studied problem, for which dozens of heuristics have been proposed in the literature to minimize makespan (i.e., the overall execution time of a workflow). For an extensive list of such heuristics, we refer to [12]. Among these heuristics, Heterogeneous Earliest Finish Time (HEFT) [13] has been one of those that are most frequently cited and used,
which is known to be simple while providing good performance. Apparently, all these heuristics are different with our study in that their scheduling does not take energy consumption into account. Nevertheless, as there is a need to produce an initial schedule to minimize makespan while assuming all tasks run at the highest frequency level, we adopt HEFT as the heuristic used in the first phase of our algorithm.

Because DVFS is a promising energy saving technique that can be incorporated into scheduling, a considerable number of scheduling heuristics based on DVFS have been developed for diverse applications and computing platforms. In [14], two power-aware scheduling algorithms are proposed to reduce the execution speed of future tasks by reclaiming the time unused by a task, so as to reduce the total energy consumption of multiprocessor systems. In [15], an analytical model of general tasks for dynamic voltage scaling is presented, and based on the model, two scaling algorithms are proposed to provide real-time guarantee with reduced capacity in a real-time system. In [16], the problem of allocating a set of independent tasks onto a real-time system consisting of dynamic voltage scaling (DVS)-enabled processors is formulated as a generalized assignment problem, and a linearization heuristic is extended for solving the problem. In [17], an adaptive energy-efficient scheduling strategy is proposed to adjust voltages according to the workload conditions of a DVS-enabled cluster to achieve a good trade-off between energy conservation and meeting real-time requirement. In [18], a three-phase DVFS algorithm is proposed to reduce energy consumption for chip multiprocessors by clustering task slack times using task graph unrolling. In [19], an energy-aware task scheduling algorithm is proposed based on stochastic model of bag-of-tasks applications on heterogeneous computing systems. These DVFS-based scheduling approaches are conducted on homogeneous computing systems [14, 18], or single-processor systems [15], or focused on independent tasks [16, 17, 19]. Therefore, they cannot address issues like task dependency and/or processor heterogeneity, which are considered in our study.

There are also DVFS-based scheduling heuristics focusing on DAG applications as well as heterogeneous systems. The work in [20] analyzes energy-efficient DAG scheduling problems with deadline constraint under various DVFS models. In [21], an online DVFS mechanism for slack prediction is used in scheduling to achieve energy saving for real complex high-performance computing applications. In [22], a linear combination of the maximum and minimum operating frequencies for task execution is proposed to decrease energy consumption. Wang et al. [23] try to extend execution time of non-critical tasks in a DAG to reduce overall energy consumption without affecting the total execution time. Energy-conscious scheduling heuristic (ECS) [24, 25] is a multi-objective optimization algorithm aiming at simultaneously minimizing the makespan and energy consumption with a low complexity. Mezmaz et al. [26] proposed a hybrid algorithm consisting of genetic algorithm and ECS. Zhou et al. [27] extended ECS with a new objective function so as to improve its performance on both makespan and energy saving. These studies differ with ours in that they focus on optimizing simultaneously on makespan and energy, other than minimizing energy with a constraint of deadline.

One of the prior studies closely related to our work is presented by Huang et al. [28, 29], who proposed an enhanced energy-efficient scheduling algorithm (EES) to reduce energy consumption while meeting performance-based service level agreement (e.g., deadline constraint). This algorithm exploited the slack room between initially scheduled tasks and re-allocated them in a global manner to achieve energy saving. It has been shown that EES outperforms naive heuristics such as greedy-based and path-based as described in [28]. The work newly presented in [11], which focuses on energy reducing of workflows with certain deadlines, is also closely related to our work. The ESFS heuristic is proposed in [11] to iteratively scale the frequency for each task only when overall energy consumption can be decreased. We regard EES and ESFS as natural competitors which will be compared with our approach in the evaluation.

3. MODEL

In this section, we describe the application model, the system model, and the energy model used in our work, and then specify the problem we are going to address.
3.1. Application model

We use a DAG to represent an application to be scheduled. In a DAG, nodes denote tasks, and edges represent data transmission between tasks. In our work, we use $G = (N, E)$ to represent a DAG, which consists of a set of nodes $N$ and a set of edges $E$. A node $i \in N$ represents the corresponded task and an edge $(i, j) \in E$ represents the intercommunication and precedence constraint between node $i$ and $j$. For an edge $(i, j)$, $i$ is called a parent node of $j$, and $j$ is called a child node of $i$. A child node cannot start execution until all of its parents has finished and all the required data transmission has arrived. Parentless nodes are called source nodes; childless node is called sink node. Apparently, an entry node of $G$ must be a source node, and an exit node is a sink node. For standardization, we specify in this paper that a DAG has only a single entry node and a single exit node. One can easily see that all DAGs with multiple entry or exit nodes can be equivalently transformed to this standardization [30]. For illustration, a simple example DAG is shown in Figure 1, where the weight attached to each edge denotes the time required to transmit the data between the connected nodes.

In order to meet the precedence constraints, the start time and the finish time of task $j$ on host $q \in N$ are computed by

$$
ST(j, q) = \max\{FT(l^*, q), \max_{k \in Par_j} \{FT(k, p_k) + TC((k, p_k), (j, q))\}\}
$$

(1)

$$
FT(j, q) = ST(j, q) + EC(j, q)
$$

(2)

where $EC(j, q)$ represents the execution time of task $j$ on host $q$; $FT(l^*, q)$ denotes the finish time of task $l^*$ which is the currently last task on host $q$; $Par_j$ represents the set of all parent tasks of task $j$; $p_k$ denotes the host which task $k$ is assigned to, and if there is no task assigned to host $q$, $FT(l^*, q)$ is equal to zero. In the case of the entry task, we have

$$
ST(entry\_node, p_{entry\_node}) = 0
$$

(3)

After the scheduling is completed, the makespan of the schedule is defined as

$$
makespan = \max_{k \in N} FT(k, p_k)
$$

(4)

3.2. System model

We consider a set of DVFS-enabled cloud hosts which are fully interconnected and equally capable of running any applications. All the hosts can run at different voltage and frequency levels. While the host is in idle, it stays at its lowest voltage and lowest frequency level for the maximal energy saving. Hereby, we assume a set of DVFS-enabled hosts (denoted by $P$) that are fully connected. It is assumed that the time needed to transmit per unit of data from one host to another, named

![Figure 1. A simple directed acyclic graph.](image-url)
transmission rate, is constant and pre-known. If task \(i\) and \(j\) are allocated to the same host, the transmission cost is assumed zero. To simplify the computation model, it is also assumed that one host can only run one task at a time and no preemption is considered.

Each host can operate in a set of voltage supply levels (VSL, denoted by \(V\)), each of which is corresponding to a specific operating frequency (denoted by \(F\)). For task \(n_i\), we assume its execution time on a host \(p\), which operates on VSL 0 (denoted by \(EC(i, p, v_{p,0})\)), is pre-known; thereby, the execution time of \(n_i\) on a different VSL \(j\) (denoted by \(EC(i, p, v_{p,j})\)) is computed by \(EC(i, p, v_{p,j}) = EC(i, p, v_{p,0}) \times F_{p,0}/F_{p,j}\).

### 3.3. Energy consumption model

We adopt the energy model used in [25], which is derived from the power consumption model in complementary metal-oxide semiconductor logic circuits. Because we assume that hosts consume a certain amount of energy while idling, the total energy consumption of the execution for a DAG is composed of the direct and indirect energy consumption. The direct energy consumption is defined as

\[
E_d = \sum_{i=1}^{n} \alpha V_i^2 F_i \Delta t_i
\]

where \(n\) is the number of tasks, \(\alpha\) is a device related constant, \(V_i/F_i\) is the voltage/frequency on which the host operates when executing task \(i\), and \(\Delta t_i\) is the amount of time taken for \(n_i\)'s execution. On the other hand, the indirect energy consumption is defined as

\[
E_i = \sum_{j=1}^{p} \sum_{d_{j,k} \in D_j} \alpha V_{j,\text{low}}^2 F_{j,\text{low}} \Delta w_{j,k}
\]

where \(p\) is the number of hosts, \(D_j\) is the set of idling slots (between time 0 and the makespan) on host \(p_j\), \(V_{j,\text{low}}/F_{j,\text{low}}\) is the lowest supply voltage/frequency on \(p_j\), and \(\Delta w_{j,k}\) is the amount of idling time for \(d_{j,k}\). Then, the total energy consumption is defined as

\[
E_{\text{total}} = E_d + E_i
\]

Note that one may argue that when a particular host finishes its all allocated tasks before the whole workflow completes, this host can be switched off for energy saving. However, this is normally not the case in cloud computing, as a user often rent a batch of hosts for a certain period of time. Therefore, for each host used to execute the workflow, we accumulate energy consumption for every idle time slot within the whole course of the execution of the workflow.

### 3.4. Scheduling problem

The scheduling problem in our study is allocating \(n\) tasks in a DAG to \(p\) DVFS-enabled cloud hosts to minimize energy consumption while still meeting a pre-specified deadline constraint. We assume all DAGs start execution at time 0 and the makespan is defined as the latest finish time of \(n\) tasks after the scheduling is completed.

### 4. METHODOLOGY

In this section, we firstly provide a simple example which motivates us to track the change of energy consumption during the course of scheduling, and then describe the details of our proposed heuristic.

#### 4.1. Motivational example

We provide in this subsection a simple example to illustrate that lowering operating frequency does not always result in a reduction of overall energy consumption.
The DAG example shown in Figure 1 is used for illustration. We assume that there are two cloud hosts $P1$ and $P2$. The execution time of every task on $P1$ is the same as on $P2$ and is presented in Table I. $P1$ and $P2$ can operate on the voltage/frequency levels specified as Pair 4 and Pair 5, respectively, in Table II. Say, the deadline is 7.9. The DAG is initially scheduled based on maximum frequency, as shown in Figure 2(a), with a makespan of 7.6.

Now let us focus on task 5 on $P1$. If the operating frequency of task 5 is lowered to 2400 while leaving other tasks unchanged, the makespan of the DAG extends to 7.9, which still meet the deadline. The new scheduling result is shown in Figure 2(b). After lowering the frequency of task 5 by this way, one can easily verify that the overall energy consumption is surprisingly increased. Lowering the frequency of task 5 does reduce the energy consumption of $P2$; however, as the makespan is extended, the energy consumption of $P1$ arises because of extra idle time. In details, according to Eq. 7 (here we assume $\alpha = 1$), the increased energy amount of $P1$ is $E_{P1}^{up} = 1081.125$, while the decreased energy amount of $P2$ is $E_{P2}^{down} = 592.2$. In sum, the increased overall energy consumption is $E_{P1}^{up} - E_{P2}^{down} = 488.925$. This finding motivates our consideration of tracking overall energy consumption change in the course of scheduling.
4.2. The GTI heuristic

Prior studies on deadline constrained energy optimization problems only consider time factors (such as slack time, deadline constraint, etc.) when making scheduling decision. This has drawbacks because expending the task execution time (which means lowering the CPU frequency) does not always cause the reduction of overall energy consumption. As the task execution time is expended, the makespan may be longer. This may introduce extra ‘idle energy consumption’ as defined in the previous section (denoted by \(E_i\)) and result in unexpected increasing energy consumption, for which the previous subsection provides a concrete example.

Noting the major drawbacks of existing heuristics, we propose a novel algorithm called GTI (Group Then Individual) to reduce energy consumption in a more sophisticated way. GTI initially schedules tasks with maximum frequencies, then rescales the frequencies for energy saving in a ‘group’ manner, and then in a ‘individual’ manner. During the rescaling, we evaluate the instant change of overall energy consumption and make scheduling decision accordingly.

As shown in Figure 3, the proposed GTI algorithm can be divided into three major phases: initial mapping, rescaling by group, and rescaling by individual.

- **Phase 1: Initial Mapping.** HEFT, as described in [13], is firstly employed to allocate the DAG to the hosts, with the maximum frequency. This determines the mapping between tasks and hosts, the execution order of tasks on the same host, and the communication time among different resources.

- **Phase 2: Rescaling by Group.** The basic idea of this phase is to evenly rescale the frequency for the tasks allocated onto the same host for global optimality. The rescaling is performed a higher frequency level to its next, which means we rescale the voltage and frequency level at most by 1. The rescaling is also performed in the fashion of host by host, namely, we rescale all tasks on the same host at a time till the phase terminates. When choosing the host for rescaling, we always prefer the one whose rescaling can obtain the maximum energy saving. In this phase, we consider the saving on direct energy only. As indirect energy consumption is closely related to the makespan, which is far away from being determined at this stage, so we use \(\Delta E_d^p\) (as mentioned in the algorithm description) to denote the change on direct energy on host \(p\) and always choose the host with maximum \(\Delta E_d^p\) for rescaling. It is worth mentioning that at this stage, it is possible that the overall energy consumption may even temporally be greater than \(E_{\text{base}}\) (which stands for the energy consumption based on scheduling all tasks on maximum frequency). This occurs due to the rising of indirect energy consumption. Therefore, we use a threshold parameter \(\theta\) for subtle control. If the new energy consumption obtained by group rescaling is not greater than \(\theta \cdot E_{\text{base}}\), then the rescaling will be approved; otherwise, the rescaling will not be considered.

- **Phase 3: Rescaling by Individual.** In this phase, the rescaling is still performed from one level to its next; however, we rescale only one task at a time till the phase terminates. We greedily look for the individual task for which the rescaling can achieve the maximum energy saving while the deadline constraint is still met. In this phase, we take into account indirect energy consumption and therefore use \(\Delta E_i^t\) (as mentioned in the algorithm description) to judge which task we should pick up at the moment we need to make a choice. This can guarantee that in this phase, the algorithm is always going on the direction on reducing the overall energy while still meeting the deadline.

4.3. Further investigation on the GTI heuristic

Although it has been shown in [10] that GTI significantly outperforms EES on energy saving, there are still two open issues regarding with the configuration of GTI which are worthy of further investigation.

- In GTI, there is a scaling phase ‘rescaling by individual’ already, is the phase ‘rescaling by group’ really necessary?
- If the ‘rescaling by group’ phase is included, how to specify the \(\theta\) value used in this phase?

We examine these two questions through evaluation in the rest of this subsection.

4.3.1. Evaluation setting. We partially adopt the same voltage–frequency setting as used in [28], which is derived from three real heterogeneous processors: AMD Turion MT-34, Pentium M
AN ADAPTIVE DEADLINE CONSTRAINED ENERGY-EFFICIENT SCHEDULING HEURISTIC

Input: A DAG application G, a set of hosts R and a deadline d.
Output: An energy efficient schedule ΩG for G.

Phase 1: Initial Mapping
1: Allocate G to R using an existing DAG scheduling heuristic (e.g., HEFT) based on the maximum frequency fmax.
2: Compute the current makespan Mcur.
3: Compute the current overall energy consumption Ecur.
4: Record the current overall energy consumption as Ebase.
5: for each task i allocated on host p do
6: Initialize Voltage Supply Level for task i, Vi,p = 0 (i.e., the maximum frequency).
7: endfor

Phase 2: Rescaling by Group
8: Group all tasks allocated on host p into Gp.
9: Set ContG to be TRUE;
10: while ContG is TRUE repeat
11: for each host p do
12: Tentatively increase Vi,p by one level for each task i in Gp.
13: Compute the new makespan Mimp,p, the overall energy consumption Eimp,p, and the saving on Ed, namely ∆Ed = Edold - Eednew, where Ed is defined in Eq.(5).
14: endfor
15: if there exists such a host p that satisfies Mimp,p ≤ deadline and Eimp,p ≤ θ·Ebase then
16: Select p* with the maximum ∆Ed,p.
17: Increase Vi,p* by one level for each task i in Gp*.
18: Update Mcur and Ecur.
19: else
20: Set ContG to be FALSE.
21: endif
22: endwhile

Phase 3: Rescaling by Individual
23: Set ContI to be TRUE;
24: while ContI is TRUE repeat
25: for each task i on each host p do
26: Tentatively increase Vi,p by one level for each task i (presumably i is allocated to host p).
27: Compute the new makespan Mimp and the saving on Et, namely ∆Et = Etold - Eetnew, where Et is defined in Eq.(7).
28: endfor
29: if there exists such a task i that satisfies Mimp ≤ deadline and ∆Et ≥ 0 then
30: Select i* with the maximum ∆Et,i.
31: Increase Vi*,p by one level (presumably i* is allocated to host q).
32: Update Mcur and Ecur.
33: else
34: Set ContI to be FALSE.
35: endif
36: endwhile
37: if the deadline is met, return the schedule with the final V setting;

Figure 3. The description of group then individual.

processor, and AMD Athlon-64. In addition, two more voltage–frequency sets were complemented. These are composed of the voltage–frequency pairs as presented in Table II.

For the evaluation, we considered two types of DAGs, which are Air Spatial Normalization (AIRSN) with 51 tasks and SDSS with 123 tasks, as shown in Figure 4(b) and (d). These DAGs are considered to be run on 3, 5, and 8 cloud hosts. Each host randomly adopt one voltage–frequency pair as presented in Table II, respectively.

In order to model task execution times, we first generate randomly task execution times in the range of [1, 100]. In brief, two values are selected from a uniform distribution in the range of [1, 10]. The product of the two selected values is computed and adopted as a generation of the mean task execution time. As for data transfer times, we randomly generate them assuming a given communication-to-computation ratio which was randomly chosen from the interval [0.1, 1].

We measure the total energy consumption during the whole period of DAG execution, which involves the task execution time and the idle periods. We generate a HEFT schedule ΩHEFT, in which
tasks are executed at maximum operating frequency while scaling the frequency to the lowest level when the host is idle. We use the energy consumption $E_{\text{base}}$ of $\Omega_{\text{HEFT}}$ as the baseline. The energy consumption measurement of all evaluated heuristics is normalized by $E_{\text{base}}$, and we take ‘energy saving ratio’ $\gamma$, which is the percentage by which a heuristic can improve the $E_{\text{base}}$, as the metric. For deadline setting, we use the makespan $M_{\text{base}}$ of $\Omega_{\text{HEFT}}$ as the lower bound. We also obtain the makespan by applying HEFT with the minimum task operating frequency $M_{\text{extra}}$. We import a parameter ‘deadline ratio’ $\bar{\beta}$ for deadline setting: $\text{deadline} = M_{\text{base}} + \bar{\beta} \times (M_{\text{extra}} - M_{\text{base}})$ Starting from 0, we increase $\bar{\beta}$ in a step of 0.1 and up to 1.

Generally, in one experiment, we firstly specify a DAG type, then generate a DAG of this type and its associated host setting. Next, $M_{\text{base}}$ is computed and a set of deadlines is obtained according to different $\beta$s. Under each deadline, we measure the energy saving ratio for each heuristic. For each DAG type, we repeat the experiment 100 times, and for each $\bar{\beta}$, the energy saving ratio for each heuristic is averaged over 100 results. Note that it is possible that one heuristic fails to meet the specified deadline, in such a case, the energy saving ratio is regarded as zero (since we can always use $\Omega_{\text{HEFT}}$ to meet the deadline).

4.3.2. Is the ‘rescaling by group’ phase necessary? In order to examine whether the ‘rescaling by group’ phase is necessary, we compare two variants of GTI: one is the GTI with the ‘rescaling by Group’ phase (denoted by ‘with group’) and the other is the opposite (denoted by ‘without group’). With the $\theta$ value fixed to be 1.01, we measure the performance of these two variants under different settings of deadlines and DAGs.

In Figure 5, it can be seen that generally, GTI with the group phase performs better than GTI without the group phase. However, this is not always the case. In the case of DAG AIRSN with five hosts, GTI without the group phase may obtain better energy saving ratio when the deadline ratio approaches to 1.0. In the case of DAG AIRSN with eight hosts, GTI without the group phase outperforms its competitor in almost all settings of deadline ratio. On the other hand, when the DAG

Figure 4. Directed acyclic graph applications used in the experiments [30].
Figure 5. Comparison of group then individual variants (with or without the ‘rescaling by group’ phase)

SDSS is used, GTI with the group phase always performs better no matter how the number of hosts and the deadline ratio vary. In summary, among six different settings of DAG and resource number, ‘AIRSN with eight hosts’ is the only one that suggests GTI should not have the ‘rescaling by group’ phase. This raises a question: what makes the case of ‘AIRSN with 8 hosts’ different with other combinations of DAG and number of hosts?

In order to analyze structure features of AIRSN and SDSS we used, we levelized these two DAGs. For example, the AIRSN DAG with 51 nodes can be divided into 11 levels, and the numbers of tasks at each level (from top to bottom) are {10,10,3,3,1,1,1,10,1,1,10}. We call each of these numbers
‘levelized task number’. Associated with the different sizes and topology structures between AIRSN and SDSS, we have the following guess: given a fixed number of hosts, when the hosts used is relatively few when comparing with the levelized task numbers of the DAG, the ‘rescaling by group’ phase is preferable. This motivates us to enable the ‘rescaling by group’ phase adaptive, that is, whether this phase should be applied will depend on the topology of the DAG and the number of hosts used.

4.3.3. Specifying the $\theta$ value. Assuming the ‘rescaling by group’ phase is applied, we compare the performance of GTI when different values of $\theta$ is used. Specifically, we let $\theta$ equals to 1.0, 1.005, 1.01, 1.02, and 1.04, respectively, and measure the energy saving ratio accordingly. For example, when the DAG AIRSN with five hosts is used, the results of energy saving ratios as deadline ratio and $\theta$ vary are shown in Figure 6(a). One can easily see that the setting of $\theta$ does significantly affect the performance of GTI, especially when the deadline ratio approaches 1.0, that is, the deadline constraint is relaxed. However, there is no single setting of $\theta$ that appears dominant to other $\theta$ settings for every setting of $\beta$. In order to compare the statistical performance of different $\theta$ values, we rank the $\theta$ values according to their energy saving ratios (from high to low) for every setting of $\beta$. For the results shown in Figure 6(a), the corresponding ranking results are collected in Figure 6(b). We compute the average rank for each value of $\theta$ over 11 settings of $\beta$ and highlight the $\theta$ value with the highest average rank. In Figure 6(b), $\theta = 1.005$ has the highest average rank of 2.0.

This evaluation and ranking procedure has been performed for each combination of different DAG types (AIRSN or SDSS) and different numbers of hosts (3, 5, or 8). The $\theta$ values with the best average rank for all DAG-resource combinations are shown in Table III.

Again, it can be seen in Table III that a good choice of $\theta$ is closely related to the size of the DAG and the number of the hosts used. As the number of hosts increases, it will be better to set $\theta$ closer to 1.0. When the size of DAG is relatively big, a larger value of $\theta$ is preferable. These observations inspire us to choose $\theta$ dynamically according to the size of DAG we are going to schedule and the number of hosts we are going to use.

4.4. The AGTI heuristic

Based on the observations gained from the previous subsection, we recognize the weakness of the initial version of the GTI heuristic and extend it to a new version named AGTI.

![Figure 6. Result of $\theta$ evaluation when directed acyclic graph AIRSN and five hosts are used.](image)

(a) Energy saving ratios as $\beta$ and $\theta$ vary  
(b) Ranking statistics of (a)

Table III. $\theta$ values with best average rank.

<table>
<thead>
<tr>
<th>$\theta$ with best average rank</th>
<th>DAG AIRSN</th>
<th>DAG SDSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 hosts</td>
<td>1.01</td>
<td>1.04</td>
</tr>
<tr>
<td>5 hosts</td>
<td>1.005</td>
<td>1.02</td>
</tr>
<tr>
<td>8 hosts</td>
<td>1.0</td>
<td>1.01</td>
</tr>
</tbody>
</table>

Copyright © 2015 John Wiley & Sons, Ltd.  
DOI: 10.1002/cpe
Considering the impact of the system and application characteristics to the performance of the heuristic, AGTI extends GTI on two facets:

- In contrast to GTI runs the ‘rescaling by group’ phase unconditionally, we make the ‘rescaling by group’ phase conditional in AGTI. Let \( R \) denote the number of hosts used, \( \text{Num}_{\text{total}} \) denote the number of levels of the DAG used, \( \text{Width}_i \) denote the levelized task number of the \( i \)th level. We count \( \text{Num}_{\text{wide}} \) that is the number of levels on which \( \text{Width}_i \times 2 > R \) is true. If \( \text{Num}_{\text{wide}} > \text{Num}_{\text{total}}/2 \), the ‘rescaling by group’ phase will be adopted by AGTI; otherwise, the phase will not be adopted.
- In contrast to GTI uses a constant as the \( \theta \) value, we dynamically compute \( \theta \) in AGTI according to the system and application characteristics. Given a DAG with \( n \) nodes, \( l \) levels and \( r \) hosts, \( \theta \) is computed by \( \theta = 1 + 0.005 \times \left[ \frac{n}{1 + \text{width}} \right] \).

5. EVALUATION

In this section, four heuristics EES, ESFS, GTI, and AGTI are compared with the same experimental setting as described in Section 4.3.1. Two more DAGs, including Montage with 34 nodes and Laser Interferometer Gravitational wave Observatory (LIGO) with 77 nodes, as shown in Figure 4(a) and (c) are used in the evaluation.

Figures 7–9 show the evaluation results for different types of DAGs when different numbers of hosts are used.

Figure 7. Energy saving ratio results with three hosts.
When three hosts are used, all evaluated heuristics can achieve significant energy saving compared with the HEFT without considering scaling operating frequency. Moreover, for almost every value of $\beta$ with each DAG type, those heuristics that consider the overall energy consumption during scheduling, including ESFS, GTI, and AGTI, perform better than EES, which only tries to lower the operating frequency as much as possible. One can easily note that the disadvantage of EES compared with other heuristics weakens as the size of DAG grows. When Montage, AIRSN, and LIGO are used, AGTI obtain better energy saving ratio than ESFS in most settings of $\beta$, except for $\beta = 0.0$. When SDSS is used, in cases where $\beta \leq 0.5$, ESFS performs a little better than AGTI, while in cases where $\beta > 0.5$, AGTI has advantage over ESFS. In the comparison between AGTI and GTI, for almost every setting of $\beta$ combined with Montage, AIRSN, and SDSS, AGTI outperforms GTI. This indicates the extension of AGTI to GTI does achieve substantial improvement. For LIGO with three hosts, as AGTI behaves the same as GTI does, the same energy saving ratio results are obtained for these two heuristics.

When five hosts are used, in all cases of $\beta$ and DAG type, AGTI, GTI, and ESFS outperform EES. Especially for Montage and AIRSN, the energy saving ratio of AGTI appears significantly better than EES. The improvement of AGTI over EES can be up to 750% (energy saving ratio 17% vs. 2%), when $\beta = 1.0$ and Montage is used. It is also interesting to observe how energy saving ratio changes as $\beta$ varies. When Montage and AIRSN are used, for EES, its energy saving ratio has an obvious trend of going down as $\beta$ grows from 0.5 to 1.0. In contrast, the energy saving ratio results of ESFS, GTI, and AGTI are relatively stable when $\beta$ changes.

When eight hosts are used, the result of SDSS looks similar to that when three or five hosts are used. However, for Montage, AIRSN, and LIGO, an ad hoc result can be observed. When $\beta$
Figure 9. Energy saving ratio results with eight hosts. EES, energy-efficient scheduling; GTI, group then individual; AGTI, adaptive group then individual.

is greater than a certain value (0.4 for Montage and AIRSN, and 0.8 for LIGO), applying EES to the schedule generated by HEFT may result in increase of overall energy consumption. This is because when there are too many hosts allocated to execute a small-size workflow, most of the hosts may be left idle during the execution of the workflow. Therefore, the ‘gain’ of energy saving from scaling frequency may not compensate the ‘loss’ of extra energy consumption caused by more idle time. As a result, EES may suffer a disaster result with an unexpected increase of overall energy consumption. This well justifies our consideration of overall energy consumption changes when making scheduling decision in our heuristic. When comparing ESFS and AGTI, one can easily see AGTI outperforms ESFS when Montage and AIRSN are used. However, when SDSS is used, ESFS performs slightly better than AGTI. In most settings of $\beta$ for Montage, AIRSN and LIGO, AGTI makes significant improvement to GTI. For SDSS, AGTI and GTI perform exactly the same.

In summary, we present several major findings as follows:

- AGTI and GTI can obtain significant energy saving no matter how the setting of deadline constraint and the number of hosts changes.
- With considering overall energy consumption during the course of scheduling, AGTI, GTI, and ESFS outperform EES in almost all cases of experimental setting. The advantage of these heuristics of EES seems correlated to the ratio of task number over host number ($\text{THR}$). The less the $\text{THR}$ is, the greater the improvement will be.
- AGTI, as an extended version of GTI, outperforms GTI in almost all cases of experimental setting. With considering the characteristics of system and application, AGTI shows slight advantage over ESFS in majority of the settings of our experiments.
The evaluation results demonstrate the drawback of considering only time factors in energy scheduling. EES can sometimes result in even higher energy consumption than applying a makespan–optimization heuristic without considering energy saving. This is illustrated by the negative energy saving ratio shown in the Figure 9. In contrast, AGTI, GTI, and ESFS never encounters such a problem.

6. CONCLUSION

Based on the fact that most existing energy-efficiency-oriented scheduling heuristics lack the consideration of tracking change of overall energy consumption when making scheduling decision, we propose a novel heuristic GTI which takes the lacked consideration into account and scales operating frequency for tasks in a more sophisticated manner. In addition, we investigate how the system and application characteristics affect the performance of GTI and proposed the AGTI heuristic, an extended version of GTI. Our simulation experiment provides promising results showing not only the significance and potential of our proposed heuristics in the reduction of energy consumption but also the superior performance over existing approaches. In the future, we hope to assessing the value of our heuristics with real DVFS-enabled hosts rather than simulated ones.

ACKNOWLEDGEMENT

This work is supported by the National Natural Science Foundation of China (NSFC, grant no. 61202361).

REFERENCES
