J Grid Computing
DOI 10.1007/s10723-015-9334-y

An Energy-Efficient Task Scheduling Algorithm in DVFS-enabled Cloud Environment

Zhuo Tang · Ling Qi · Zhenzhen Cheng · Kenli Li · Samee U. Khan · Keqin Li

Received: 30 November 2014 / Accepted: 31 March 2015

Abstract The growth of energy consumption has been explosive in current data centers, super computers, and public cloud systems. This explosion has led to greater advocacy of green computing, and many efforts and works focus on the task scheduling in order to reduce energy dissipation. In order to obtain more energy reduction as well as maintain the quality of service by meeting the deadlines, this paper proposes a DVFS-enabled Energy-efficient Workflow Task Scheduling algorithm: DEWTS. Through merging the relatively inefficient processors by reclaiming the slack time, DEWTS can leverage the useful slack time recurrently after severs are merged. DEWTS firstly calculates the initial scheduling order of all tasks, and obtains the whole makespan and deadline based on Heterogeneous-Earliest-Finish-Time (HEFT) algorithm. Through resorting the processors with their running task number and energy utilization, the under-utilized processors can be merged by closing the last node and redistributing the assigned tasks on it. Finally, in the task slacking phase, the tasks can be distributed in the idle slots under a lower voltage and frequency using DVFS technique, without violating the dependency constraints and increasing the slacked makespan. Based on the amount of randomly generated DAGs workflows, the experimental results show that DEWTS can reduce the total power consumption by up to 46.5 % for various parallel applications as well as balance the scheduling performance.

Keywords Cloud computing · DVFS · Energy saving scheduling · Heterogeneous · Heuristic algorithm

Z. Tang (✉) · L. Qi · Z. Cheng · K.L. Li · K.Q. Li
College of Information Science and Engineering,
Hunan University, Changsha 410082, China
e-mail: ztang@hnu.edu.cn

K. Li
e-mail: lik@newpaltz.edu

S. U. Khan
Department of Electrical and Computer Engineering,
North Dakota State University, Fargo, ND 58102, USA
e-mail: samee.khan@ndsu.edu

K. Li
Department of Computer Science, State University of
New York, New Paltz, NY 12561, USA

1 Introduction

Large-scale businesses and scientific applications, which are usually composed of big-data, multitasking, time-variant, and fluctuating workloads [1, 2], have become the mainstream of current technologies. For instance, Hadoop has been combined by Amazon with cloud computing called Amazon EMR, which provides cloud service for people that allows users quickly and easily handle large amounts of data. With the rapid growth of data scale, using the cloud

Published online: 19 April 2015

computing technology to deal with diverse applications has become increasingly important. Cloud computing [3], with virtualization [4] as the key enabling technology, provides an elastic scaling-up and scaling-down provisioning mechanism. Realization of these techniques are based on the large scale of cloud data centers. And the high price of energy consumption has become a critical issue for these data centers.

During the last few years, the high price of energy consumption has become a critical issue [5]. One of the research estimates that a data center with 50,000 computing nodes may use more than one hundred million kwh/year [6], equivalent to the electricity consumption for a 100,000 population urban in one year. The energy consumption in data centers will still increase quickly in the next ten years [7]. On the other hand, the CPU utilization for severs is comparatively not high. The work in [8] shows that the average CPU utilization of more than 5,000 servers during a six-month period is between 10 and 50 percent of their maximum utilization levels. These researches illustrate the PUE (Power Usage Effectiveness: all energy use of data center and IT load consumption ratio) of most datacenters are unsatisfactory. Obviously, the criterion to evaluate mechanisms for parallel applications only focus on minimizing the schedule length, but rarely meeting the growing advocacy for green computing system. This paper considers that efficient task scheduling in cloud environment should not only try to obtain a minimal completion time but also increase the system resource utilization as well as reduce the energy consumptions.

Based on the green computing concept, developing energy-efficient mechanisms for parallel applications becomes increasingly attractive. The problems of parallel application scheduling are NP-hard in the general case. Most of the static scheduling problems can be solved by an application represented as Directed Acyclic Graph (DAG) scheduling, similar to the work of Braun [9], in which nodes stand for application tasks and edges represent intertask data dependencies. Moreover, various other mechanisms for reducing the energy consumption have been investigated in the earlier works, such as Dynamic Voltage/Frequency Scaling (DVFS) [10] and Dynamic Power Management (DPM) [11]. DPM turns the idle components off leading the resources to the hibernate mode to reduce the power consumption. While it only works when the idle time is long enough, DVFS has been proven to be a very promising technique with its demonstrated capability for energy savings [12–15]. It is based on the fact that energy consumption in CMOS circuits has a direct relationship with the square of the supplied voltage and frequency [16, 17], a large reduction in power consumption can be achieved by switching between processor's voltages/frequencies during task execution while guaranteeing some performance. However, most of these approaches are confronted with the fact that combining optimum to each sub-problem may ignore the global optimality for the crucial system performance. In addition to DVFS technique, if applications are not time-critical, we can consider minimizing the number of used processors by taking advantage of the idle processor time among the running tasks in parallel to increase the resource utilization. In this way, users may need to tolerate a little delay of execution for decreasing system energy consumption. Thus, finding the inefficient processors and turning them off combining the DVFS technique may be a promising approach to reducing energy dissipation as well as guaranteeing the performance.

In this paper, based on meeting the performance-based service level agreement, we propose a new energy aware scheduling algorithm named DVFS-enabled Efficient-energy Workflow Task Scheduling (DEWTS) to optimize the energy savings through DVFS technique for parallel applications in the heterogeneous distributed computing systems. In this paper, the effect and performance of DEWTS are estimated through comprehensive experiments, under the maximum performance conditions, different number of processors, various extension ratios, different values of CCR, and different degree of parallelism. And the evaluating indexes are four performance metrics: energy consumption ratio (ECR), system resource utilization ratio, average execution time, and energy saving ratio. The main contributions of this paper are summarized below.

1. This paper proposes an energy-aware task scheduling algorithm. Within a given deadline, this algorithm can distribute the parallel applications in workflows to appropriate processors, and deals with them at the appropriate time slots to reduce energy consumption as well as meeting the required performance.

2. Numerical experiments are given to verify that DEWTS can increase the CPU utilization of processors and reduce significant amount of energy consumption in a wide range of workflow structures compared with other researches.
3. We analyze the factors which are affecting the performance of our algorithm.

The remainder of this paper is organized as follows. We compare our work with related research effort in Section 2, including several different scheduling heuristics on heterogeneous systems, power estimation and optimization techniques, and some energy aware scheduling algorithms. Section 3 describes the energy models, cloud application, and system used in this paper. In Section 4, we present the details of our scheduling algorithm DEWTS, and illustrates a simple case to explain this algorithm better. The experimental results and evaluation analyses are presented in Section 5. Finally, Section 6 concludes the whole paper.

2 Related Works

2.1 Task Scheduling in Heterogeneous Environment

Due to the NP-complete nature of the parallel task scheduling problem in general cases [18, 19], many heuristics have been proposed in recent researches [15] to deal with this problem, and most of them achieve good performance in polynomial time. Static task-scheduling algorithms can be classified into two main groups (see Fig. 1), heuristic-based and guided random-search-based algorithms.

In the previous works, the heuristic-based algorithms can be classified into a variety of categories, such as list scheduling algorithms, clustering heuristics, and duplication-based algorithms. Among them, the list scheduling algorithms are generally more practical, and their performances are better at a lower time complexity. A list scheduling algorithm maintains a list of all tasks of a given graph according to their priorities. It has two typical phases: task priority phase for selecting the highest-priority ready task, and processor selection phase for deciding suitable processors to minimize the predefined cost function (which can be the execution start time). Some notable achievements are obtained in recent years [20–23]. Among these algorithms, Heterogeneous-Earliest-Finish-Time (HEFT) algorithm [21] is a well-known heuristic list-scheduling which has an $O(e \times q)$ time complexity for e edges and q processors. It has two major processes: task prioritization and processor selection with insertion-based scheduling policies.

Clustering heuristics are researched to select the tasks among which there are relatively large traffic into a group in distributed environment [33]. In these algorithms, the tasks in same group will be distributed to the same processor to decrease the inner communications among the computing nodes. A typical clustering heuristics algorithm is DSC (Dominant Sequence Clustering) [24], whose basic idea is to put critical path scheduling tasks to the same processor, and start them at the earliest time. Analogously, for decreasing

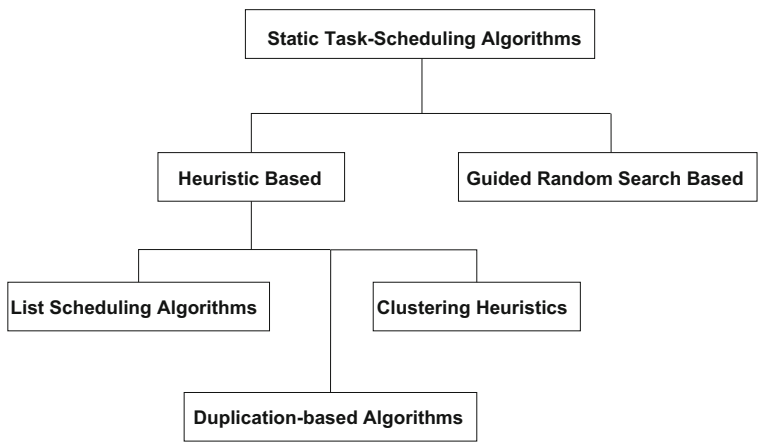


Fig. 1 Classification of Static task-scheduling algorithms

the overall makespan, duplication-based algorithms are proposed to replicate the forerunners using the idle slots in processors, which can reduce the interprocess communication distinctly. HLD (Heterogenous Limited Duplication) [25] is a typical duplication-based algorithm, which provides a way to make appropriate choices in heterogeneous environments when copying the predecessor tasks.

2.2 Energy-Saving Optimization

In recent yeas, much attention has focused on energy aware scheduling for single processor[26], homogeneous system [27, 28], and heterogeneous resources [15, 17, 29, 30]. Many efficient techniques have been researched for reducing the energy consumption, such as DVFS mentioned in Section 1, based on which, there have been a significant amount of task scheduling works. In DAG scheduling model, each task has an earliest start time (EST) and an earliest finish time (EFT) respectively. For specific tasks, the range between EST and EFT usually larger than the actual execution time, and we call the difference between them as slack time. For the running process of a system, amount of slack time are usually produced while waiting the output of predecessor tasks, or executing a task with earlier completion before its deadline. Slack time reclamation technique is adopted in much of recent researches. Kim et al. [14] proposed a power aware scheduling algorithm of bag-of-tasks applications with deadline constraints on DVFS-enabled cluster systems. In [31], for reclaiming the slack time slots to save energy, Kimura et al. provided a slacking algorithm for adjusting the frequency of CPU dynamically to extended the task execution time.

Lee et al. [22] presented two energy-conscious scheduling (ECS and ECS + idle) heuristics which took account into the balance between makespan and energy consumption for parallel tasks in heterogeneous distributed computing systems. In [17], Huang et al. designed a way to lower the frequency of non-critical tasks for parallel applications in heterogeneous distributed computing systems, and reassigned the tasks to appropriate time slots to low power consumption, named Enhanced Energy-efficient Scheduling (EES) algorithm. The goals of these above works are to minimize the energy consumption of the tasks while still meeting the performance based on the determined service level agreement (SLA). Nevertheless, this approach may not perform well in dealing with communication intensive applications, and most of them do not lead to global optima with energy consumption and time cost.

Except for the above works, most other researches only focus on either lessening the completion time or reducing the energy consumption. The objectives of most existing scheduling algorithms are to shorten the schedule length without caring about the energy consumption. Different from the researches aforementioned, our scheduling algorithm aims at reducing the energy consumption by decreasing the number of inefficiently processors. Meanwhile, through combining DVFS technique with list-based task scheduling polices, this algorithm can retain the quality of service by meeting the deadlines given by the providers.

3 Models

3.1 System Model

In this work, we assume that the target system consists of a set heterogeneous processors: $P = \{p_i\}$, each one is connected in a fully interconnected topology. We presume that the set of task graphs is $N = \{n_i\}$. We also presume that computation can be overlapped with communication, which means data can be transmitted from one processor to another while a task is being executed on the recipient processor. Each processor $p_j \in P$ is DVFS enabled which means that it can be operated at different voltage levels and clock frequencies. For each processor $p_j \in P$, we define the supply voltage sequence set as $V = \{v_s\}$, and the clock frequency set as $F = \{f_s\}$. While the supply voltage operates at level v_1, the clock frequency will operate in level f_1. Since the machine still consumes energy while under the idle state, it will stay at its lowest voltage state v_{lowest} for maximum energy saving. In this paper, we will ignore the overheads of the frequency transitions for they take a negligible amount of time (e.g., 10us-150us [15]).

3.2 Cloud Application Model

Generally, parallel workflow applications can be represented by a directed acyclic graph (DAG) as shown in Fig. 2. The task graph $G = (N, E)$ consists of a set of vertices N and a set of edges E, where N is the set of n tasks partitioned from an application, and E is the set of edges between the tasks which represents the precedence constraints. Each $edge(i, j) \in E$ between task n_i and n_j also represents inter-task communication. Namely, task n_j can not start until task n_i has transmitted its output.

A task without predecessors is called an entry task n_{entry} (such as n_0 in Fig. 2a), and a task without successors is called an exit task n_{exit} (such as n_5 in Fig. 2a). If there are more than one entry tasks (such as n_0, n_1, n_2 in Fig. 2b), more than one exit tasks (such as n_3, n_5 in Fig. 2b), then it needs to introduce a virtual entry task (such as n_{00} in Fig. 2c) or a virtual exit task (such as n_6 in Fig. 2c), which will connect multiple entrance tasks or exit tasks. This process makes the DAG graph has one and only one entrance or exit task. The virtual entry (exit) task is a zero-cost node which is connected to all the real tasks with zero-cost edges, that does not affect the tasks schedule.

The weight on a task n_i labeled as w_j represents the computation cost. In a heterogeneous computing environment, the computing time may be different even on the same processor due to various jobs. If a task n_i runs on the processor p_j, we denote its computation cost as $w_{i,j}$. In this way, the average executioncost of a task n_i on all available processors is defined as (1):

$$\overline{w_i} = \frac{\sum_{j=1}^{p} w_{i,j}}{p} \tag{1}$$

We denote the weight on an edge as $c_{i,j}$, which represents the communication cost between task n_i and n_j. When both tasks n_i and n_j are allocated to the same processor, $c_{i,j}$ becomes zero for we assume that the intra-processor communication cost can be ignored. The data transfer rates between processors are stored in matrix B with size $p \times p$. The communication costs of processors are given in a p-dimensional vector S. In addition, task executions of a given application are assumed to be non-preemptive which is possible in many systems. And $data_{i,j}$ represents the data size transferred from task n_i to n_j. The communication cost between task n_i (scheduled on p_m) and n_j (scheduled on p_k) is defined as (2):

$$c_{i,j} = S_m + \frac{data_{i,j}}{B_{m,k}} \tag{2}$$

Before scheduling, average communication costs are used to label the edges. The average communication cost between task n_i and n_j is defined as (3):

$$\overline{c_{i,j}} = \overline{S} + \frac{data_{i,j}}{\overline{B}} \tag{3}$$

where $\overline{B}$ is the average transfer rate among the processors, $\overline{S}$ is the average time cost of communication

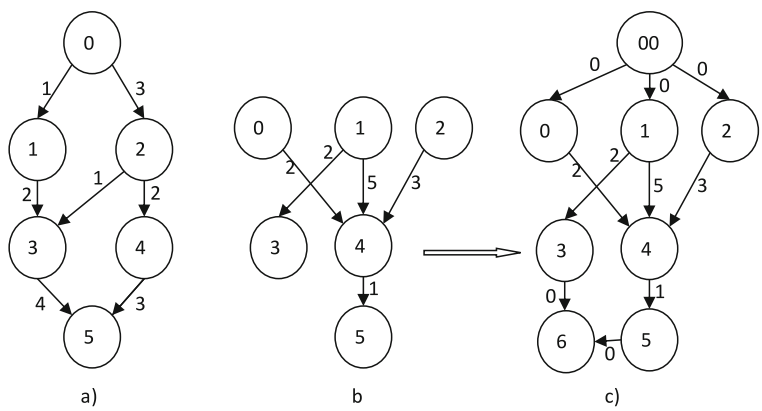


Fig. 2 A simple task graph

startup. Tasks are ordered in our algorithm by their scheduling priorities which are based on upward ranking [21]. The upward rank of a task n_i is recursively defined as (4):

$$rank_u(n_i) = \overline{w_i} + max_{n_j \subset succ(n_i)} \left(\overline{c_{i,j}} + rank_u(n_j)\right) \tag{4}$$

Without loss of generality, we use $EST(n_i, p_j)$ and $EFT(n_i, p_j)$ to denote the earliest start time and the earliest finish time of task n_i which been scheduled on processor p_j respectively. For the entry task n_{entry}, the EST can be calculated as (5):

$$EST(n_{entry}, p_j) = 0 \tag{5}$$

For the other tasks in the graph, starting from the entry task, the EST and EFT values can be calculated as (6) and (7):

$$EST(n_i, p_j) = \begin{cases} 0, & if\ n_i = n_{entry} \\ \max\left\{avil[p_j], \max_{n_m \subset pred(n_i)} \left(AFT(n_m) + c_{m,i}\right)\right\}, & otherwise \end{cases} \tag{6}$$

$$EFT(n_i, p_j) = EST(n_i, p_j) + w_{i,j} \tag{7}$$

where $avil[p_j]$ is the earliest finish time of the last assigned task n_k of processor p_j. $pred(n_i)$ is the immediate predecessors set of task n_i, $pred(n_i) = \{\forall j | \exists (j \rightarrow i), i \in N, j \in N\}$. And $AFT(n_m)$ represents the actual finish time of $task(n_m)$. $max\left\{avil[p_j], max_{n_m \subset pred(n_i)} \left(AFT(n_m) + c_{m,i}\right)\right\}$ returns the time when all data needed by n_i has arrived at processor p_j. $AST(n_m)$ presents the actual start time of $task(n_m)$. If $n_m = n_{entry}$, the $AST(n_m) = EST(n_m) = EST(n_{entry}) = 0$, then we can get $AFT(n_m)$ by calculating $AFT(n_m) = AST(n_m) + t_m$ recursively.

After all tasks in a graph are scheduled, the schedule length will be the actual finish time of the exit task n_{exit}. We call the longest path of the scheduled task graph as the critical path (CP) and the finish time of the latest task as the schedule length or makespan. If there are more than one exit tasks, the makespan MS of the latest task can be defined as (8):

$$MS = \max\{AFT(n_{exit})\} \tag{8}$$

3.3 Energy Model

The power consumption of CMOS logic circuits for an application are composed of dynamic and static energy consumption: $E_{dynamic}$ and E_{static}. Because the most expensive and time-consuming part is the dynamic power dissipation [11], static energyconsumptions are ignored in this paper. Dynamic power dissipation $P_{dynamic}$ can be defined as (9):

$$P_{dynamic} = K \times v_{j,s}^2 \times f_s \tag{9}$$

where K is a constant parameter related to dynamic power, depending on the capacities of devices. $v_{j,s}$ denotes the supplied voltage at level s on the processor p_j, and f_s denotes the frequency with the matching $v_{j,s}$. Based on this, the total energy consumption when machines working can be defined as (10):

$$E_{busy} = \sum_{i=1}^{n} K \times v_{i,p_{j,s}}^2 \times f_{p_{j,s}} \times t_{i,j} = \sum_{i=1}^{n} P_{dynamic,i} \times t_{i,j} \tag{10}$$

where $t_{i,j}$ is the execution time of task n_i on processor p_j, and $v_{(i,p_{j,s})}$ represents that task n_i is scheduled on the processor p_j under voltage s. Moreover, $f_{p_{j,s}}$ denotes the frequency of processor p_j with voltage level s. For the supplied voltages and frequencies cannot be adjusted to zero during the idle periods of processors, the voltage has to be at the lowest state v_{lowest} to save the most energy, the energy consumption of idle periods for all available processors can be defined as (11):

$$E_{idle} = \sum_{j=1}^{p} k \times v_{jlowest}^2 \times f_{jlowest} \times t_{jidle} = \sum_{j=1}^{p} p_{jidle} \times t_{jidle} \tag{11}$$

where $v_{jlowest}$ and $f_{jlowest}$ are the voltage and frequency of the processor p_j under the lowest voltage respectively, and t_{jidle} denotes the idle time of p_j.

Based on the above formulations, total energy consumption of a DAG application can be defined as (12):

$$E_{total} = E_{busy} + E_{idle} \tag{12}$$

4 Efficient Energy Scheduling Algorithm

As the overall scheduling processes in DEWTS, Algorithm 1 aims at minimizing the schedule length and energy consumption as much as possible. DEWTS has three major phases: initial task mapping phase, processors merging phase, and task drawing phase. To reduce the number of processors being used, the appropriate time slots are firstly picked out to place the tasks which come from other low-utilized processors. And then the ready tasks can be scheduled on DVFS-enabled processors to reduce energy consumption whenever the tasks have slack time. In this section, each phase is illustrated and analyzed in detail.

Algorithm 1 DEWTS(DVFS-enabled Efficient energy Workflow Task Scheduling)

Input:
$G(N,E)$: A DAG graph of the input tasks;
P: A set of DVFS-enabled processors.
Output:
Schedule tasks of $G(N,E)$ into processors of P.
//Calculate the value of makespan with HEFT;
1: *MS_Computing*();
//Processors merging phase;
2: *Processors_Merging*();
//Task slacking phase by using DVFS technique;
3: *T_Slacking*();
4: Calculate the energy consumption;
5: **return** the assignment of tasks to lower voltage/frequency processors.

Initial Task Mapping Phase This phase requires to obtain the priorities of all tasks in descending order according to $rank_u$ through traversing a DAG upward by starting from the exit task to the entry task. Some researches call it b-level sorting [15]. In this process, we firstly just need sorting one time for we just need to get a simple scheduling order without violating the dependency constraints among tasks. And then, the well-known heuristic list-scheduling algorithm HEFT is used to calculate the initial makespan MS of the latest task on the list. Finally, based on MS, meanwhile according to a user given extension ratio α, $\alpha \geq 0$, we can calculate the overall allowed time D of the all tasks according to (13):

$$D = MS \times (1 + \alpha) \tag{13}$$

Our objective is to reduce energy consumption as much as possible in accomplishing all the tasks under the condition of maintaining deadline D. Algorithm 2 shows the implementation details of calculating the initial makespan MS.

Algorithm 2 *MS_Computing*()

Input:
A DAG $G(N,E)$;
A set P of DVFS-enabled processors;
The extension ratio α.
Output:
Makespan MS and the deadline of all tasks in DAG $G(N,E)$;
1: Calculate the b-level value $rank_u(n_i)$ by traversing the DAG upward with starting from the exit task;
2: Allocate the given tasks to all available processors with the HEFT algorithm;
3: Calculate the makespan MS according to Eq. (8);
4: Calculate the deadline D according to Eq. (13);
5: Mark the DAG as unexecuted;
6: **return** makespan MS& deadline D.

Processors Merging Phase As shown in Algorithm 3, we firstly calculate the number of assigned tasks for each turn-on processors, and then sort the processors in descending order according to $\{p_1, p_2, \ldots, p_n\}$ based on $rank_m$. If $rank_m$ values of two different processors are equal, the processors with smaller energy utilization(p_{eu}) should be placed behind. The calculative process p_{eu} of corresponding processors is shown as (14):

$$p_{j_{eu}} = \frac{\left(\sum_{i=1}^{rank_m(p_j)} w_{i,j}\right) \times p_{j_{max}}}{\left(\sum_{i=1}^{rank_m(p_j)} w_{i,j}\right) \times p_{j_{max}} + \left(MS - \sum_{i=1}^{rank_m(p_j)} w_{i,j}\right) \times p_{j_{idle}}} \tag{14}$$

where $p_{j_{max}}$ denotes the maximum power dissipation of processor j, and $p_{j_{idle}}$ denotes the power dissipation when processor j is idle.

Algorithm 3 *Processors_Merging*()

Input:
A DAG: $G(N,E)$;
A set of DVFS-enabled processors: P;
The b-level;
The MS and D obtained from Algorithm2.
Output:
Get the optimized turn-on processors list.
1: Set $MS' = MS$;//record the initial scheduling length asMS length as MS';
2: // Merge the processors until the scheduling length larger than the deadline;
3: //initialize sequence;
4: **for** each $i \in [1,n]$ **do**
5: Do $rank_i = 0$;
6: **end for**
7: **while** $MS' \leq D$ **do**
8: Calculate the number of tasks each turn-on processor has been assigned, denoted as $rank_m$;
9: **for** each $j \in [1,n]$ **do**
10: **for** each $i \in [1,n]$ **do**
11: **if** task n_i is scheduled on processor j **then**
12: Do $t_{p\_j} += w_{i,j}$;
13: $rank_j$++;
14: **end if**
15: **end for**
16: **end for**
17: For tie-breaking $rank_m$, caculate the energy utilization of corresponding processor $p_{j_{eu}}$ as: $p_{j_{eu}} = \frac{t_{p\_j} \times p_{j_{max}}}{t_{p\_j} \times p_{j_{max}} + (MS - t_{p\_j}) \times p_{j_{idle}}}$;
18: Assumes that the processor's $rank_m$ is lower when p_{eu} is smaller;
19: Sort the processors$\{p_1, p_2, ..., p_{k-1}\}$ in descending order of $rank_m$;
20: Set k as the number of the processors which have been assigned tasks;
21: Run tasks list on processors $\{p_1, p_2, ..., p_{k-1}\}$ based on HEFT algorithm and gain the new value of MS;
22: // The unused processors can be shutdown only if the scheduling length is not longer than the deadline;
23: **if** $MS' \leq D$ **then**
24: Turn off the processor p_k from P;
25: $P' = P - \{p_k\}$;
26: $k = k$-1 ;
27: $p = k$;//There new number of available processors is k;
28: $MS' = MS$;// Save the scheduling length in MS' temporarily;
29: Mark the DAG as unexecuted;
30: **end if**
31: **end while**
//Obtain the effective processors set as P';
32: **return** P'.

This algorithm will repeat scheduling the tasks on the first $k-1$ processors until the total scheduling length is larger than D, where k is the number of processors with initial arrangement tasks on the first phase. After completing a circuit, if the scheduling length is still no larger than D, shutdown the processor which has not been assigned jobs, and the value k minus 1. After these steps, we can store the last scheduling results as our final processors selection, and mark the surviving processors as a set of P'. When finishing this step, for dealing with a group of given tasks in a workflow, the relatively efficient processors can be reserved to reduce the waste of energy consumption.

Time Slacking Phase In this phase, the idle time slots can be slacked and reassigned using DVFS technique without violating precedence constrains. As shown in Algorithm 4, for a specific task, the latest allowable finish time LFT should be calculated as (15):

$$LFT(n_i) = \begin{cases} D, & if\, n_i = n_{exit} \\ min_{n_j \in succ(n_i)} \left(LFT(n_j) - t_j - c_{i,j} \right), & otherwise \end{cases} \tag{15}$$

where $succ(n_i) = \{\forall j | \exists (j \rightarrow i), i \in N, j \in N\}$ formalizes the direct successor node of task n_i, which t_j denotes its execution time. The allowable slack time of task n_i can be calculated as (16):

$$Slack(n_i) = LFT(n_i) - EST(n_i) - t_i \tag{16}$$

The next is to lower and optimize the clock frequency of task n_i. Similarly to the EES algorithm [17], we first choice the job n_i with the largest LFT. If $Slack(n_i) > 0$, compare the EST of task n_i with the LFT of the previous task on the same processor. If $LFT(n_i) > EST(n_x)$, it shows that there are overlaps of the slack times between these two tasks, repeat this step forward until finding a task n_m which has no slack overlapping time slots with subsequent task n_{m-1}. Equation (17) calculates the total execution time from task n_i to n_m on processor p_j:

$$T_{run} = t_i + t_x + \cdots + t_m \tag{17}$$

And (18) calculates the total idle time on processor p_j from n_i to n_m:

$$T_{total} = AFT(n_i) - EST(n_m) \tag{18}$$

Then, the ideal smallest operating frequency f'_{n_i,p_j} for task n_i can be calculated by (19):

$$f'_{n_i,p_j} = f_{p_j,0} \times max\left(\frac{t_i}{t_i + Slack(n_i)}, \frac{T_{run}}{T_{total}} \right) \tag{19}$$

Through the comparing between f'_{n_i,p_j} and the set of voltage/frequency levels of processor p_j, we can pick out the nearest value $f_{p_j,s}$ as the actual operating frequency of task n_i, $f_{p_j,s} \geq f'_{n_i,p_j}$. Then we can set the actual operating frequency to f_{n_i,p_j}, $f_{n_i,p_j} = f_{p_j,s}$, and update task n_i's execution time to t'_i according to (20):

$$t'_i = \frac{t_i \times f_{p_j,0}}{f_{n_i,p_j}} \tag{20}$$

Through the above processes, execution time slot of task n_i on processor p_j can be changed on the range of $[LFT(n_i) - t'_i, LFT(n_i)]$. In this way, after completing scaling the frequencies, we can calculate the specific scheduled time of task n_i, and update the LFT for both task n_x and its predecessor which executing just before n_i on p_j, if it exists. The same process is repeated until all tasks are optimized.

Algorithm 4 *T_Slacking*()

Input:
Task slacking by using DVFS technique;
The scheduling results of Algorithm 3: MS', D, EST, every task's execution time.
Output:
A new executing tasks list with proper voltage/frequency pairs.

```
1: Calculate every task's LFT;
2: Calculate every task's slacking time Slack(n_i);
3: while there are un-optimized tasks do
4:    if The slacking time of task n_i Slack(n_i) exist then
5:       Select the task n_i with the largest LFT;
6:       Calculate n_i 's ideal operating frequency f'_{n_i,p_j};
7:       Pick out n_i 's actual operating frequency f_{n_i,p_j};
8:       Update the frequency of task n_i from f_{p_j max} to f_{n_i,p_j};
9:       Update the execution time of task n_i to t'_i;
10:      Assign time slot of task n_i to [LFT(n_i) − t'_i, LFT(n_i)];
11:      Update the LFT of all predecessor task for task n_i;
12:      Update the LFT of the task n_x which be executed just
         before task n_i on p_j;
13:   end if
14: end while
15: return an executing tasks list.
```

Comparing to EES [17], our algorithm ignores the step of distributing the slack time between the original makespan and the deadline to each task evenly, it only shifts time slots in the scaling slack time. This is because after the processors merging phase, the makespan MS' is near upon the deadline D, may even equal to D. On the other hand, the ideal frequency f'_{n_i,p_j} may not precisely equal to the presetting voltage/frequency levels, so we should pick out the nearest $f_{p_j,s}$ to replace f'_{n_i,p_j} without violating the initial condition: $f_{p_j,l} \geq f'_{n_i,p_j}$.

In DEWTS algorithm, while holding the overall performance of the task scheduler in the deadline given by the user, the total energy consumptions of the system are also reduced. The main idea is to optimize the number of processors used firstly, reassigned tasks on the light load processors to others, achieve the goal of reducing the number of running processors. To take advantage of residual idle slot between tasks on processors, we further use DVFS technology to reduce voltage and clock frequency of processors, and effectively extend the task execution time. The ultimate goal is to reduce the processors energy costs. The following is an example to verify the advantage of DEWTS Algorithm over HEFT, DVFS and EES.

A Simple Example To illustrate that decreasing the number of processors in conjunction with DVFS technology can improve resource utilization, and reduce total energy consumption of systems more effectively, an simple example is provided to verify the feasibility of DEWTS. To simplify the description, the example assumes that there are five isomorphic processors with DVFS function. The five processors can run in the following voltage levels {1.2v, 1.1v, 1.0v, 0.9v, 0.8v, 0.7v} within the frequency levels {1.0Ghz, 0.8Ghz, 0.6Ghz, 0.5Ghz, 0.4Ghz, 0.3Ghz}.

Firstly, we sort the tasks through b-level sorting. For the sample DAG of Fig. 3, Table 1 shows the computation costs of the ten tasks on processors. For

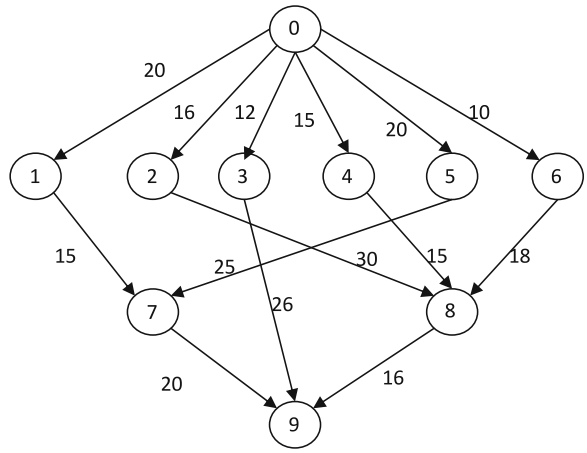


Fig. 3 A simple task graph with 10 tasks.

Table 1 The tasks list in the DAG of Fig. 3

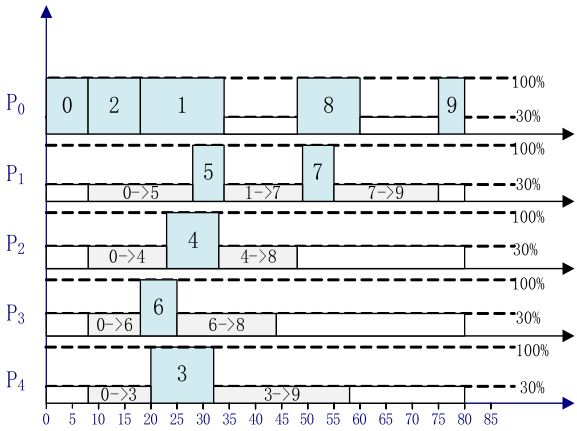
Task	0	1	2	3	4
t_i	8	16	10	12	10
Task	5	6	7	8	9
t_i	6	7	6	12	5

Table 2 Task priorities in the DAG of Fig. 3

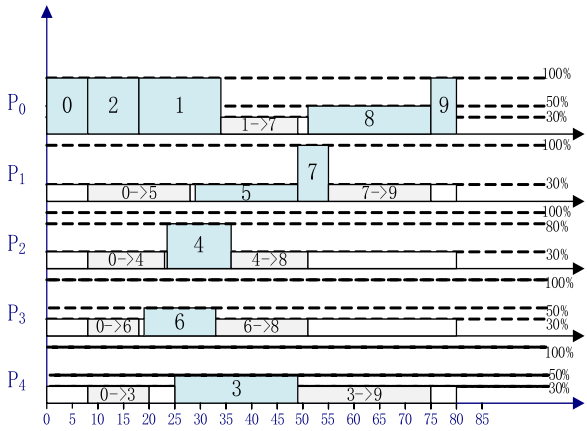
n_i	0	1	2	3	4
b-level	97	62	73	43	58
n_i	5	6	7	8	9
b-level	62	58	31	33	5

simplicity, we suppose that all these five processors have no difference with their performance. That is to say the same task have the same execution time on all of the processors.

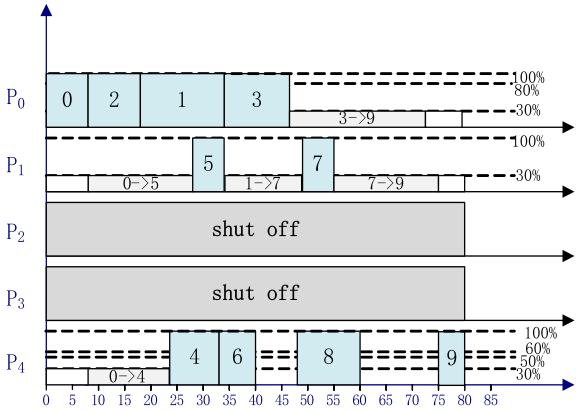
Table 2 gives the upward ranks calculated by using (4) for the given DAG. By comparing the values, we can get the order of the tasks is: $\{n_0, n_2, n_1, n_5, n_4, n_6, n_3, n_8, n_7, n_9\}$.



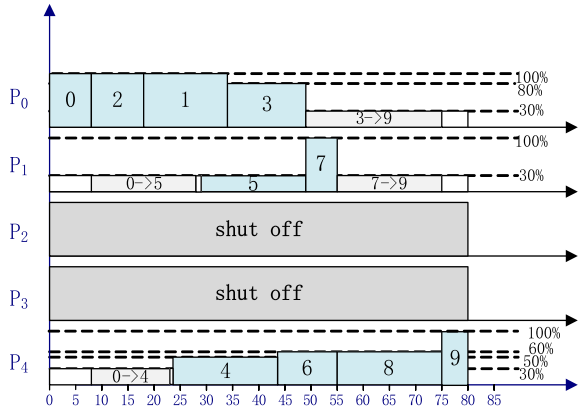
(a) Scheduling of G(N,E)in Fig. 3 with HEFT



(b) Scheduling of G(N,E)in Fig. 3 with EES



(c) Merging processors phase of DEWTS



(d) Scheduling of G(N,E)in Fig. 3 with DEWTS

Fig. 4 An example of DEWTS compared with EES and HEFT

As an illustration, Fig. 4a shows the initial schedules obtained by the HEFT algorithm for the DAG in Fig. 3, and Fig. 4b presents the results without processor merging using EES algorithm. Through Algorithm $Processors\_Merging$, we can decide which processors can be turned off. Figure 4c shows the process and result of processor merging. In this example, the processor p_3 and p_2 are turned off in turn not only without violating precedence constrains but also without increasing the schedule length. And the tasks 4 and 6 in p_2 and p_3 are all scheduled to the processor p_4. As shown in Fig. 4d, these task executing time are all slacked under lower voltages to decrease the energy consumption. It is the final scheduling result obtained by DEWTS algorithm. For simpleness and easy analysis, in this example, we assume that there are five homogeneous processors and $\alpha = 0$, in other words, the deadline D equals to MS.

According to Fig. 4a, we can see the MS with HEFT algorithm is 80, we also can easily calculate that the busy time is 92, the idle period is 308, so the utilization ratio of CPU is only 23 %, and the E_{total} is 177.756 which consists of 132.48 dynamic and 45.276 idle energy consumption. Likewise, the utilization ratioes of CPU with EES and DEWTS are 34.875 % and 54.875 % respectively, meanwhile the energy saving ratioes are 22.6 % and 29.5 % respectively. This example demonstrates that we can benefit from combining processor merging technique with DVFS technique to reduce processor energy consumption more efficiently while still meeting the performance requirement.

5 Experiments and Analysis

In this section, to evaluate the performance of our proposed approaches, we present the comparative evaluation of DEWTS with two heuristics algorithms: HEFT [15] and EES [17]. HEFT is a well-known algorithm without considering energy cost, yet proven performing well for task scheduling. EES is an enhanced energy-efficient scheduling based on DVFS technique, which can be used to decrease the frequencies of non-critical jobs in a global manner, and reassign the tasks to appropriate time slots to get low power consumption. With HEFT and EES are as the benchmarks in this paper, we choose CloudSim simulator as our experiment platform. CloudSim is a widely used framework for modeling. It can be used to simulate the cloud computing infrastructures and services, which can offer a repeatable and controllable experimental environment, as well as do not need to pay much attention to the hardware details [32].

The experiment comparisons of the algorithm are based on the following four performance metrics:

Energy Consumption Ratio (ECR) In this paper, the main performance measure of the algorithms is ECR. ECR refers to the ratio of the total energy consumed by the task execution DAG and consumption of tasks executing in the fastest finished processor on the critical path. For a specific task, the ECR can be calculated based on (21):

$$ECR = \frac{E_{total}}{K \times \sum_{n_i \in CP} min_{p_j \in P}\{w_{i,j}\} \times max_{v_{j,k} \in V_j}\{v_{j,k}\}^2 \times max_{f_{j,k} \in F_j}\{f_{j,k}\}} \tag{21}$$

System Resource Utilization Ratio The system resource utilization ratio is the percentage of used resources compared with total resources. This metric is the basic feature that we need to considered in this paper as it can direct reflect the resource utilization efficiency of an algorithm.

Average Execution Time The execution time of an algorithm is its running time for obtaining the output schedule of a given task graph. Among all the three algorithms, the one who get the minimization average execution time is the one most practical implementation.

Energy Saving Ratio In this paper, the total energy consumption can be measured during the whole period, which involves the task execution time and the idle periods. Energy saving ratio means the energy

saved of all the algorithms compared with E_{base}. In order to facilitate comparison, we use the energy consumption of HEFT schedule as the base line E_{base}, which adjusts the frequency to the lowest level when the processor is in idle. The performances of all the algorithms are measured in terms of the normalized total consumption.

5.1 Experimental Settings

Four groups of DVFS-enabled heterogeneous processors are simulated for our studies. Table 3 shows their voltage/frequency pairs and types of chosen processors.

In this experiment, we take the randomly generated DAG graphes as the tasks set for our experiments. The simulation parameters [21] depending on several characteristics are given as below.

- The number of random DAG tasks: {20, 40, 80, 160, 320, 400}.
- The set of parallelism factor β: {0.2, 0.5, 1.0, 2.0, 5.0}. A high β will lead to a DAG with shorter length but high parallelism.
- The communication to computation ratio (CCR) set: {0.1, 0.5, 1.0, 2.0, 5.0}. if CCR value is very low, it can be considered as a computation-intensive application, otherwise, it can be considered as communication-intensive application.
- The set of processors available to use is from 2 to 32, incrementing by the power of 2.
- The extension ratio α in our experiments ranges from 0 to 180 %.

To avoid biasing toward a particular algorithm, we assign several input parameters, and choose each parameter from a wide set to generate diverse DAGs with various characteristics. The experimental results are the average of the values obtained from 600 different graphs for each set of the above parameters.

5.2 Results and Analysis

This paper designs five experiments to provide the performance comparisons in different number of processors, various extension ratio, different CCR, and different degree of parallelism. The following experiments provide a detailed analysis for each group.

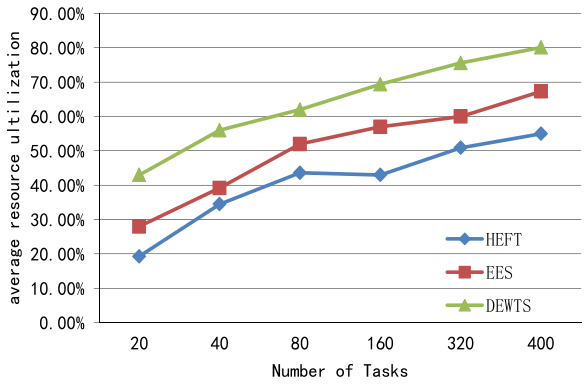
5.2.1 Estimate in the Maximum Performance Conditions

In the first set of experiments, to evaluate the method that combines the merging processors with the DVFS technique can perform better than the HEFT while considering energy consumption, we just set the extension ratio $\alpha = 0$ (see Fig. 5), namely without extending the schedule length, and other parameters are randomly selected. For each number of tasks, we iterate the experiment for 25 times, and the final results are averaged. The number of processors in this experiment is 32, and there are 8 processors for each type.

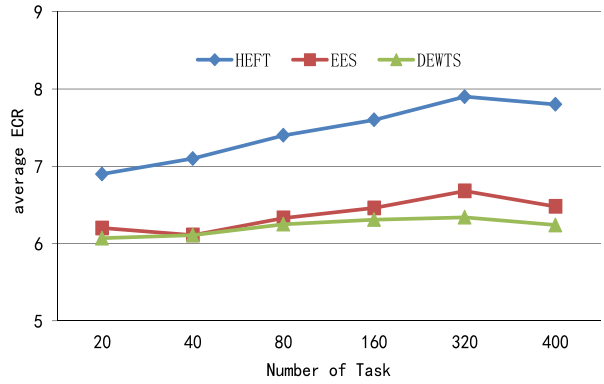
Figure 5a shows the average resource utilization for all algorithms. Obviously, we can observe that both EES and DEWTS can increase the system utilization on different levels, but DEWTS performs more

Table 3 The voltage/frequency pairs

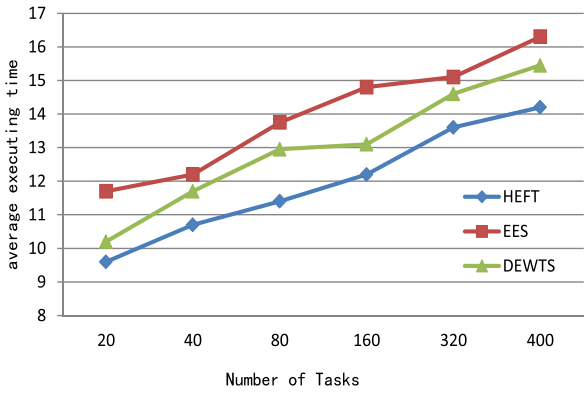
Level	AMD Athlon-64		Intel Pentium M		AMD Opteron 2218		AMD Turion MT-34	
	Voltage (V)	Frequency (GHz)	Voltage (V)	Frequency (GHz)	Voltage (V)	Frequency (GHz)	Voltage (V)	Frequency (GHz)
0	1.5	2.0	1.484	1.4	1.30	2.6	1.20	1.8
1	1.4	1.8	1.463	1.2	1.25	2.4	1.15	1.6
2	1.3	1.6	1.308	1.0	1.20	2.2	1.10	1.4
3	1.2	1.4	1.180	0.8	1.15	2.0	1.05	1.2
4	1.1	1.2	0.956	0.6	1.10	1.8	1.00	1.0
5	1.0	1.0			1.05	1.0	0.90	0.8
6	0.9	0.8						



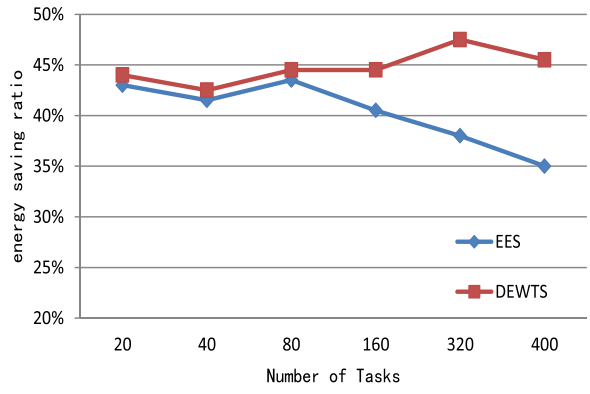
(a) Average resource utilization.



(b) Average ECR.



(c) Average running time.



(d) The energy saving ratio.

Fig. 5 Evaluation of random DAG without extension schedule length

competitively. The max utilization ratio even reaches to 80 %.

The average ECR is given in Fig. 5b. As we know from the definition of ECR, the lower ECR and the better scheduling algorithm. Figure 5b demonstrates that DEWTS has more advantages in energy saving visually, and the proportional reduction of ECR values in DEWTS is much more steadily than that in EES because of EES just considering the DVFS mechanism. From Fig. 5b, we can reach a conclusion that both DEWTS and EES perform better in large scale of workloads.

Figure 5c depicts the three algorithms for the average execution time of the task sets. The average execution time is calculated from the total execution time divided by the number of tasks. With the increasing number of tasks, the average execution time of three basic scheduling algorithms all tended to decrease. This is because with the increase number of the tasks, the scale of the random pattern DAG is also increased. It can be seen from the HEFT curve that the average execution time for a single task tends to be more stable, and gradually achieve a balance. And the average time of EES and DEWTS algorithms are significantly higher than HEFT, because the first two algorithms aim at saving energy, and both of them have varying degrees of task execution time stretching. Because DEWTS algorithm completes the processor number optimization before using DVFS technique to reduce the generation of luxury consumption, the average task execution time of DEWTS algorithm is lower than EES. Actually, EES only focuses on the energy consumption optimization, and it relatively ignores the impacts of some other conditions on performance.

Figure 5d gives the energy saving ratio for the other two algorithms compared with HEFT algorithm. From the results we can know that DEWTS algorithm saves more energy than EES as compared with EES, for DEWTS holds the step of processors merging. While increasing the number of tasks, the performance of

DEWTS becomes much more better, and the highest energy saving ratio nearly reaches up to 47.5 %. But once the number of tasks exceeds a threshold value, the energy saving ratio of these two algorithms will decrease and become equal finally. On average, the DEWTS can achieve higher energy saving ratio than EES up to 7 % from our experimental results.

From the above four figures, we can draw such conclusion: at the highest performance conditions, DEWTS and EES algorithm have shown a good energy saving effect when working set is relatively on a large scale, the energy optimization efficiency of DEWTS is slightly more obvious.

5.2.2 Estimate in Different Number of Processors

For the efficiency comparison, the number of processors used in our experiments varied from 2 to 32, incrementing by the power of 2. In this application the number of tasks is fixed, as large-scale task is more meaningful and typical than small-scale task in a distributed system, so we set the size of task as 400. Other parameters are the same as in the above experiments.

Figure 6a shows the data of average resource utilization in different number of processors for the three algorithms. The trend of data reveals that the change of processors number has less effect on HEFT, as this algorithm only focuses on reducing the scheduling time. With the increase number of processors, the running time decreases but more resources are wasted because some processors remain empty waiting, it leads to a degree of wastage. EES and DEWTS perform much better than HEFT, because they all have the step of task slacking by using DVFS technique. By comparing all the performances of the three algorithms, DEWTS has the highest resource utilization. With the increasing of processors number, this advantage becomes more obvious (the biggest gap is 0.25 compared with HEFT) as the processors merging played a crucial role.

Comparisons of average ECR under different processors are given in Fig. 6b. From this figure we know that when processor number is 2, all of the three algorithms meet their lowest ECR. Based on our experiments, their lowest average ECR values are 6.5, 6.17 and 6.1. These values are increased slowly when the number of processors is less than 8 as the total consumptions of energy increased not so fast, compared with the situation when the number of processors is larger than this value. But value of ECR in our algorithm increased much slower than others as the energy consumption is much lower.

Comparisons of average running time are given in Fig. 6c. Obviously, with the increasing of processor numbers, the average running time will decrease. When the number of processors is 8, we meet the first and the last turning point. The executing time of all the three algorithms tend to be stabilized with the increasing of processors number. The average executing time of both EES and DEWTS are higher than HEFT, because EES and DEWTS all aim at saving energy, and both of them have different degrees of stretching the execution time of a single task. Comparing these two algorithms, EES only focuses on reducing energy consumption, but DEWTS finishes processor number optimization before using DVFS technique. Hence, the average task execution time of DEWTS is naturally lower than EES.

Figure 6d shows the energy saving ratio of EES and DEWTS comparing to HEFT algorithm. Although the resource utilizations of all the three algorithms are relatively high, DEWTS seems the best algorithm. From the results, we can conclude that the energy saving ratio is small when the number of processors is less than 8. With continuously increasing the number of processors, the energy saving ratio is obviously higher. From the figure we know that the highest saving ratio of the two algorithms are up to 48 % and 62 %.

By analysing all the four experiment results, it can be concluded that increasing the number of processors can cut down the running time, which will bring the waste of resources. Therefore, in practice we cannot blindly increase the number of processors. In this experiment case, 8 processors is the best choice.

5.2.3 Estimate in Various Extension Ratios

In this experiment, we compare the performance with respect to various extension ratios. Because in a distributed system environment, large-scale task is more meaningful, so in this set of experiments we select 400 tasks with 48 available processors, namely 12 processors for each type. Other parameter values are randomly selected from the set of parameters.

Figure 7a shows the resource utilization of the three algorithms under different extension ratios. This figure depicts that both EES and DEWTS can increase the resource utilization on different levels but HEFT

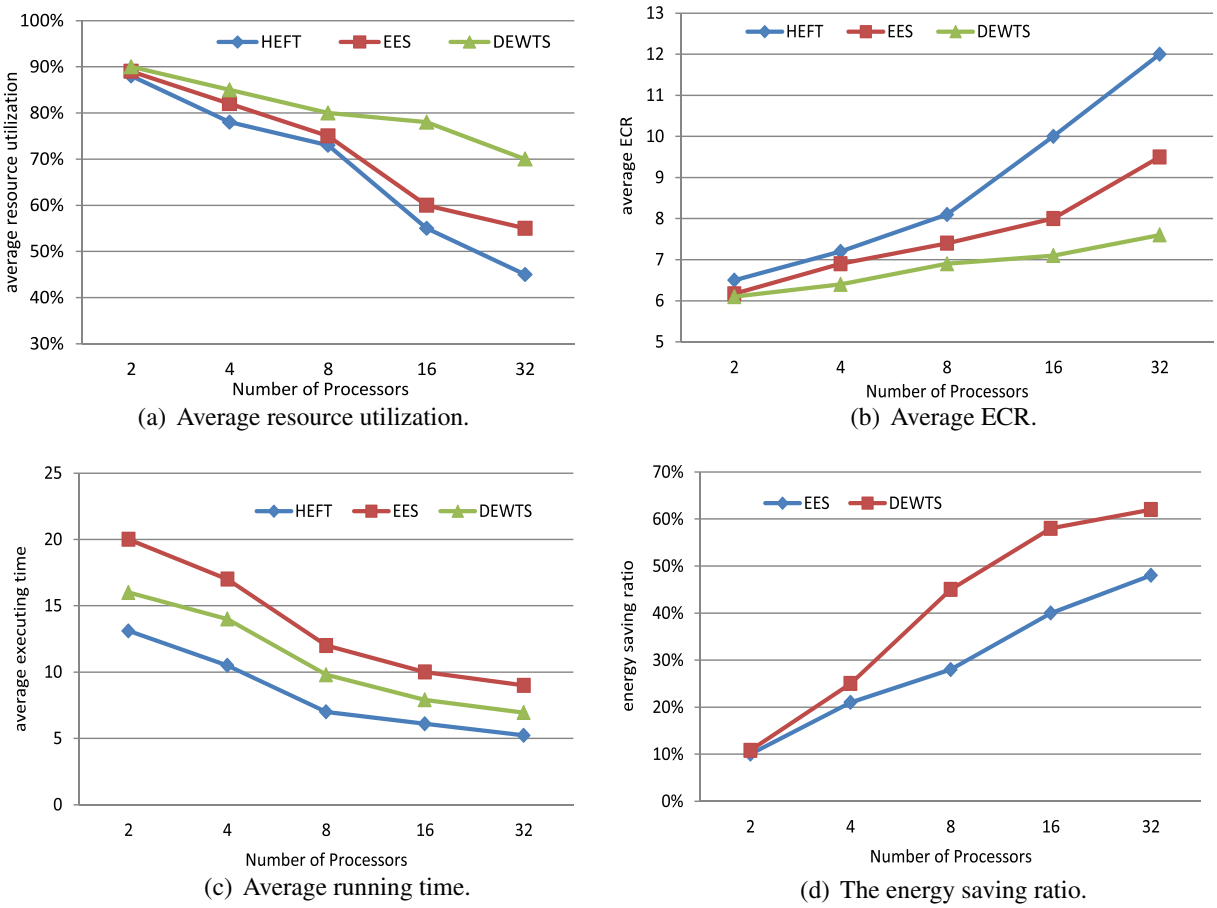
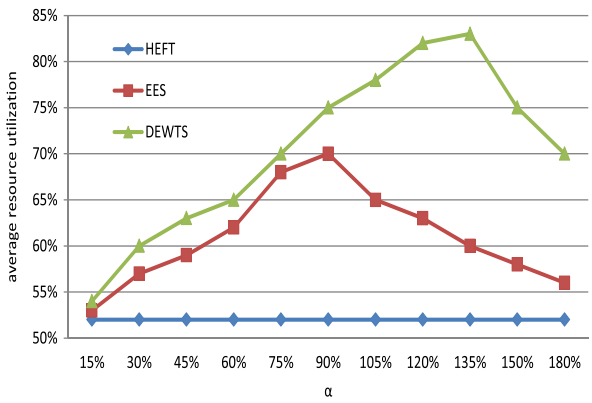


Fig. 6 Evaluation of random DAG with different number of processors

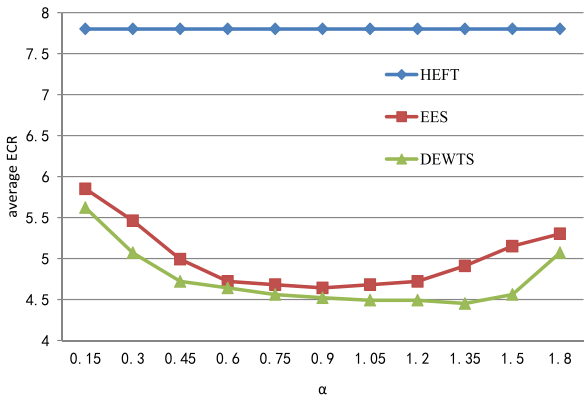
doesn't. The change of extension ratio has no effect on the performance of HEFT, for HEFT algorithm itself only focuses on how to deal with the tasks within their earliest finish time. For increasing deadline will not affect the HEFT scheduling policies, the value remains unchanged. In contrast, the extension ratio makes more effect on both EES and DEWTS. For EES, the value of resource utilization increases from 53 % to 70 % while α increases from 15 % to 90 %. In addition, the resource utilization will be decreased with the further increasing of α, because the slacking time reaches to its finite value. For DEWTS, the critical value of α is 135 %, this value comes later because of processors merging. The resource utilization value of DEWTS varied from 54 % to 83 %, increased much rapidly than EES. So we can conclude that for both EES and DEWTS, with the appropriate sacrifice of the whole tasks makespan, the resource utilization will improve greatly.

From the results shown in Fig. 7b, we can see that the extension ratio has no effect on the ECR value of HEFT. The reasons are explained in the previous paragraph. In contrast, the extension ratio has more effect on both EES and DEWTS, the ECRs decrease rapidly when α is in the range of [15 %, 60 %]. And the decrease ratio of DEWTS is faster than EES, because DEWTS has merged some processors before scaled the frequency. We can also see that the ECR of EES achieves its smallest value when α equals to 90 %, but 135 % to DEWTS. That is to say the idle slots have been reclaimed more sufficiently.

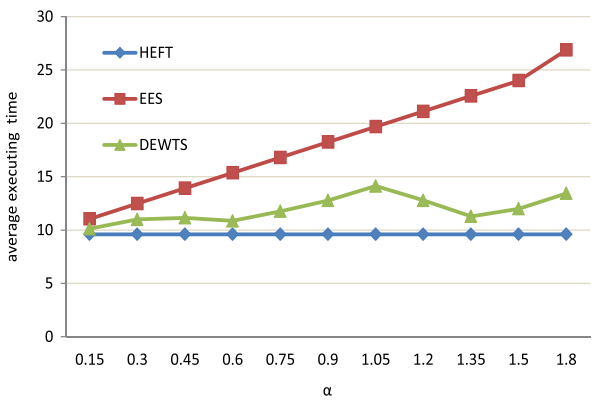
To evaluate the efficiency comprehensively, we consider the average execution time for each task as a comparison metric as well, see in Fig. 7c. The average running times for EES and DEWTS keep raising rapidly along with the scale of α, but the ratio is much lower, and the average execution time is approaching to the HEFT. Based on these results, we can say



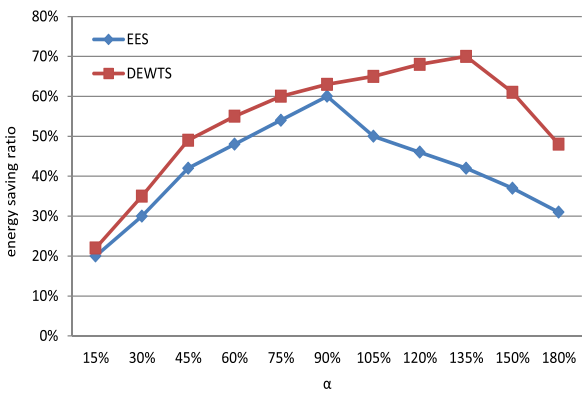
(a) Average resource utilization.



(b) Average ECR.



(c) Average running time.



(d) The energy saving ratio.

Fig. 7 Evaluation of random DAG with different extension schedule length

that EES sacrifices too much performance to meet the energy consumption. And DEWTS has a better trade-off value between the performance and reducing energy dissipation.

At last of this set of experiments, we turn our attention to study the relationship between energy saving ratio and various extension ratio. From Fig. 7d, we know that both EES and DEWTS perform better in energy saving, but the advantage of DEWTS is much more obvious as we can see from the figure, and its energy saving ratio increases more rapidly than EES. Figure 7d shows another information that if α outstrips a threshold (e.g., 90 % for EES and 135 % for DEWTS), the energy saving ratio starts decreasing after reaching the critical value, because the abilities of both processors merging and the slacking time are limited. As a periodically conclusion, DEWTS outperforms EES in terms of extending the schedule length.

From this group of experiments, we come to the conclusion that changing extension ratio has no effect on the performance of HEFT, but it does have a big effect on both EES and DEWTS. The performance of four features would be sequential improved with the increasing of α, until α outstrips a threshold (e.g., 90 % for EES and 135 % for DEWTS). Due to the features of the DAG model and the slack room for workloads are always finite, there exist an upper bound in reducing the power consumption.

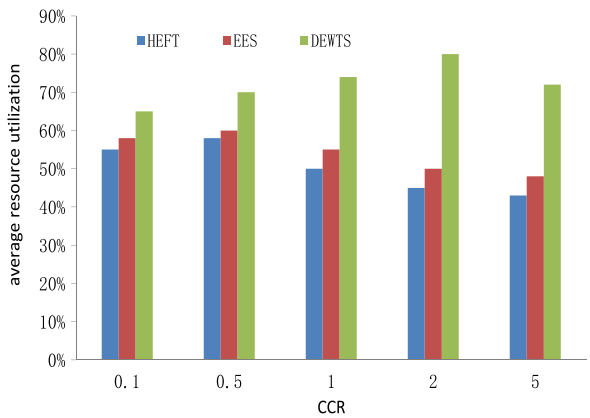
5.2.4 Estimate in Different Values of CCR

Like many previous researches, this paper also take consideration of the evaluation for CCR. This group of experiments are designed to investigate the impacts that the attributes and structure of DAG graph itself have on the three algorithms: EES, HEFT, and DEWTS. In order to avoid the effect of other factors

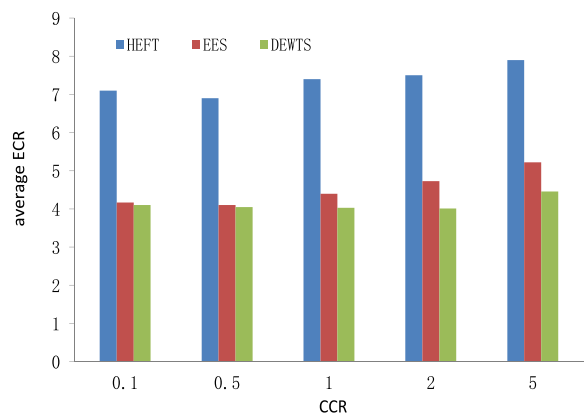
on EES and DEWTS, we set the value of the extension ratio as 100 %. The reason for this choice is because when the $\alpha = 1.0$, the ECR of EES and DEWTS are close to the minimum. Without loss of generality in distributed environment, in this set of experiments, we select 400 tasks with 48 available processors, namely 12 processors for each type. Other parameter values are randomly selected from the set of parameters.

The results of testing the average resource utilization under different CCR are given in Fig. 8a. Based on the observations from the figures, we can find that DEWTS is able to achieve quite considerably resource utilization while meeting the deadline. When the value of CCR is 2, DEWTS reaches its highest resource utilization of 80 %, which is a quite considerable value compared with EES (equal 60 % when CCR is 0.5) and HEFT (equal 58 % when CCR is 0.5). According to the results, it can be speculated that both EES and HEFT can be considered as computation-intensive applications because when CCR is low they can get their highest resource utilizations. Meanwhile, DEWTS can deal with both computation-intensive applications and communication-intensive applications.

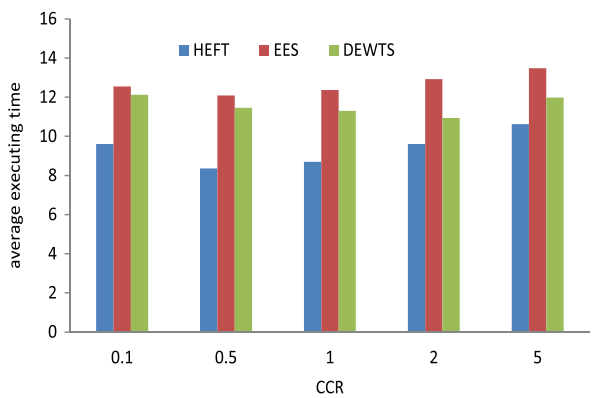
Figure 8b shows the effects of different CCR on the ECR of these three algorithms. This figure shows that when CCR is 0.5, both HEFT and EES meet their lowest ECR: 6.9 and 4.1 respectively. Meanwhile, DEWTS gets its lowest ECR (equal to 4.01) when the value of CCR is 2. Compared with HEFT, both EES and DEWTS perform much better. But compared with EES, the value of CCR has smaller impact on the ECR of DEWTS, for the fluctuation 0.5 is much less than 1.12. This phenomenon demonstrates that both HEFT and EES are suited to computation intensive applications better than communication intensive parallel



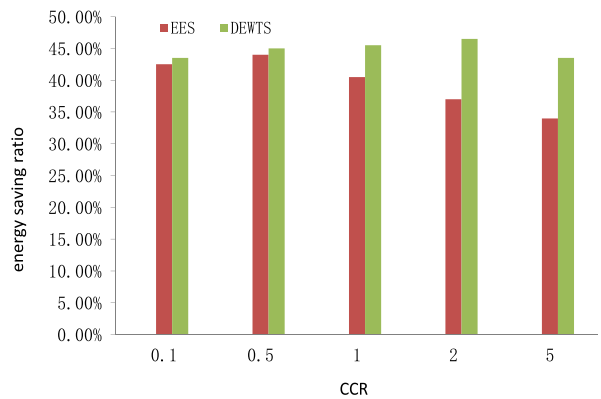
(a) Average resource utilization.



(b) Average ECR.



(c) Average running time.



(d) The energy saving ratio.

Fig. 8 Estimate in Different CCR

applications. And DEWTS can deal with both types of applications better than EES.

The average executing time is given in Fig. 8c. As HEFT has been proven to perform very competitively with a low time complexity, and extension ratio has no effect on it, so it is the best algorithm among the three as far as the average running time is concerned. As from the two above experiments results analysis, CCR has an unignored impact on those three algorithms. The average task execution time of DEWTS is lower than EES, that is because DEWTS decreases the number of processors before using DVFS technique, which brings the energy consumptions shorten.

Figure 8d describes the saving ratioes of EES and DEWTS under different CCR compared with HEFT. The results presented show that, when CCR is 0.5, the EES comes to its best saving ratio. Based on our experiments, the average saving ratio can reach up to 44 %. We can find out that DEWTS comes to the best scene when CCR is equal to 2, and the energy saving ratio is up to 46.5 %. In contrast, when CCR is equal to 2, EES can reduce 37 % of energy dissipation. And when CCR is equal to 0.5, the best saving ratio can be up to 44 %.

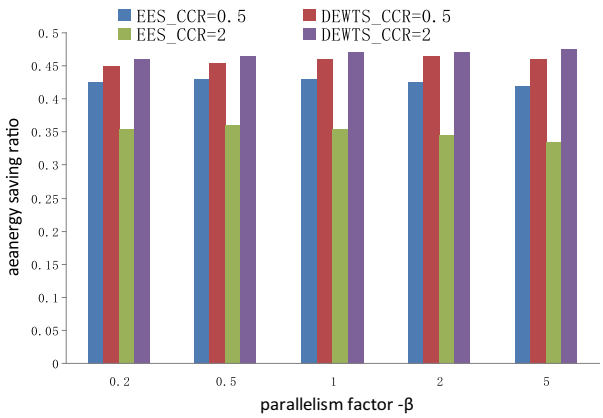
Combined with all the four experiments, we can conclude that both EES and HEFT can be appropriate for computation-intensive applications. And when CCR is low, they can get their best performances. As the advantages, DEWTS can deal with both computation-intensive and communication-intensive applications compared to EES and HEFT.

5.2.5 Estimate in Different Degree of Parallelism

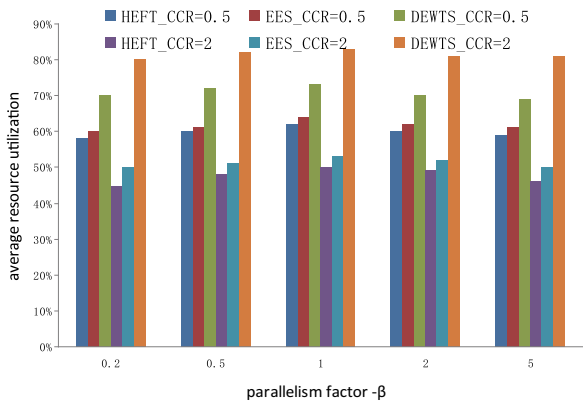
In the last experiment, for verifying the varieties of distributed system, the effects of the different degree of parallelism β are considered. And we only test the effect does the degree of parallelism has on resource utilization and energy saving ratio. To balance both EES and DEWTS, we use two CCR values (0.5 and 2.0) in this experiment, as the effects of the degree of parallelism have on ECR and average running time are similar to CCR. In order to avoid the effect of other factors on EES and DEWTS, we set the value of the extension ratio as 100 %. In this group of experiments, we select 400 tasks with 48 available processors, namely 12 processors for each type. Other parameter values are randomly selected from the set of parameters.

From Fig. 9, we can notice that although there is little impact on energy consumption by parallelism factors, DEWTS can achieve more energy saving ratio when β equals to 5 and CCR = 2, because in these conditions, there are more effective idle phases when DEWTS leverages resource utilization by merging the number of processors. Therefore, it can achieve higher resource utilization and saves more power consumption.

With DEWTS, the number of running processors are less than the compared algorithms without violating the dependency constraints. Meanwhile, the task executing time are slacked under the lower processor voltages to decrease the energy consumption. Based



(a) Average resource utilization.



(b) Average ECR.

Fig. 9 Estimate in Different parallelism factor

on all these experiments, with different types of DAG task set, DEWTS can meet the deadline given by the user constraints, not only maintains the good performance, but also reduce the idle cost and extravagant energy consumption.

There are 4-8 Voltage Identification (VID) pins for each central processing unit, the basic principle for CPU to adjust the voltage is to change the voltage on VID pins. It can generate a group of VID identification signal by changing the voltage of these identification feet. We can use voltage regulation model to realize this operating. In current processor productions, CPUs like Intel XScale and AMD Duron all support a set of voltage levels, such as AMD Mobile K6, its voltage range is from 0.9V to 2.0V, and the voltage of Intel PXA250 can be also adjusted in the range 0.85V - 1.3V. Based on these above processors, a DVFS-enabled private cloud environment can be established through an open source VM management platform CloudStack. This cloud platform can be deployed on a cluster consists of physical servers using the above DVFS-enabled processors.

6 Conclusion and Future Works

Energy consumption reducing in cloud centers is critical for green computing. This paper provides an energy-saving scheduler DEWTS based on dynamic voltage/frequency scaling algorithm. DEWTS is applicable to the scheduling system of most data centers consist of DVFS-enabled processors. Comparing to previous works, the tasks can be distributed in the idle slots under a lower voltage and frequency, without violating the dependency constraints and increasing the slacked makespan. This algorithm can be applied to various parallel applications on heterogeneous environment. It can obtain significant energy reduction as well as maintaining the quality of service by meeting the pre-set deadlines.

In future work, the system reliability will be considered. Some detail settings would be taken into account to fit the experiments to the real environment. For instance, the communication overhead, the voltage/frequency switching overhead and other uncertain parameters in the actual presence of a heterogeneous environment will be considered in the further researches.

Acknowledgments This work is supported by the Key Program of National Natural Science Foundation of China (Grant No. 61133005, 61432005). National Natural Science Foundation of China (Grant Nos. 61370095), and Open Foundation of State Key Laboratory of Software Engineering(SKLSE2012-09-18).

References

1. Cao, J., Jarvis, S.A., Saini, S., Nudd, G.R.: GridFlow: Workflow management for grid computing. In: Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, Tokyo, Japan, 198–205 (2003)
2. Tan, W., Fan, Y., Zhou, M.C.: A petri net-based method for compatibility analysis and composition of web services in business process execution language. IEEE Trans. Autom. Sci. Eng. **6**(1), 94-106 (2009)
3. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., et al.: Above the clouds: A Berkeley View of Cloud Computing, Technical Report No. UCB/EECS-2009-28, University of California, Berkerley, CA (2009)
4. Barham, P., Dragovic, B.: Xen and the Art of Virtualization, Proc. of 19thACM symposium on Operating Systems Principles, Bolton Landing, NY, USA, pp. 164-177 (2003)
5. Koomey, J.G.: Growth in data center electricity use 2005 to 2010., http://www.analyticspress.com/datacenters.html. August 1, 2011
6. Greenberg, A., Hamilton, J., Maltz, D.A., et al.: The cost of a cloud: Research problems in data center networks. ACM SIGCOMM Comput. Commun. Rev. **39**(1), 68–73 (2008)
7. Song, J.: Energy-efficiency model and measuring approach for cloud computing. J. Softw. **23**(2), 200–214 (2012)
8. Barroso, L.A., Holzle, U.: The case for energy-proportional computing. IEEE Comput. Soc. **40**(12), 33–37 (2007)
9. Braun, T., Siegel, H., Beck, N., Boloni, L., Maheswaran, M., Reuther, A., Robertson, J., Theys, M., Yao, B., Hensgen, D., et al.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **61**(6), 810–837 (2001)
10. Zhu, D., Melhem, R., Childers, B.R.: Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. IEEE Trans. Parallel Distrib. Syst. **14**, 686–700 (2003)
11. Venkatachalam, V., Franz, M.: Power reduction techniques for microprocessor systems. ACM Comput. Surv. **37**(3), 195–237 (2005)
12. Zhang, Y., Hu, X., Chen, D.: Task scheduling and voltage selection for energy minimization. In: Proceedings of 39th Design Automation Conference, pp. 183-188 (2002)
13. Rizvandi, N.B., Taheri, J., Zomaya, A.Y.: Some observations on optimal frequency selection in DVFS-based energy consumption minimization. J. Parallel Distrib. Comput. **71**(8), 1154–1164 (2011)
14. Kim, K., Buyya, R., Kim, J.: Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters. In: Proceedings of the 7th IEEE International Symposium on Cluster Computing and the

Grid, pp. 541-548. IEEE Computer Society Washington, DC, USA (2007)
15. Lee, Y.C., Zomaya, A.Y.: Energy conscious scheduling for distributed computing systems under different operating conditions. IEEE Trans. Parallel Distrib. Syst. **22**, 1374–1381 (2011)
16. Wang, L., Von Laszewski, G., Dayal, J., Wang, F.: Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS. In 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), 2010, pp. 368-377. IEEE (2010)
17. Huang, Q., Su, S., Li, J., Xu, P., Shuang, K., Huang, X.: Enhanced energy-efficient scheduling for parallel applications in cloud. In: 12th IEEE/ACM International Symposium Cluster, Cloud and Grid Computing (CCGrid), 2012, pp. 781–786 (2012)
18. Ullman, J.D.: Np-complete scheduling problems. J.Comput. Syst. Sci. **10**, 384–393 (1975)
19. Garey, M.R., Johnson, D.S.: Computers and intractability: A guide to the theory of NP-completeness, pp. 238-239. W.H. Freeman and Co. (1979)
20. Daoud, M.I., Kharma, N.: A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **68**(4), 399–409 (2008)
21. Topcuouglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
22. Lee, Y.C., Zomaya, A.Y.: A novel state transition method for metaheuristic-based scheduling in heterogeneous computing systems. IEEE Trans. Parallel and Distrib. Syst. **19**(9), 1215–1223 (2008)
23. Zomaya, A.Y., Ward, C., Macey, B.S.: Genetic scheduling for parallel processor systems: Comparative studies and performance issues. IEEE Trans. Parallel Distrib. Syst. **10**(8), 795–812 (1999)
24. Yang, T., Gerasoulis, A.: DSC: Scheduling parallel tasks on an unbounded number of processors. IEEE Trans. Parallel Distrib. Syst. **5**(9), 951–967 (1994)
25. Bansal, S., Kumar, P., Singh, K.: Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs. J. Parallel Distrib. Comput. **65**(4), 479–491 (2005)
26. Zhong, X., Cheng-Zhong, X.: Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee. IEEE Transactions on Computers **56**, 358–372 (2007)
27. Wang, L., Von Laszewski, G., Dayal, J., Wang, F.: Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS. In: 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), 2010, pp. 368-377. IEEE (2010)
28. Kyong Hoon, K., Buyya, R., Jong, K.: Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters. In: 7th IEEE International Symposium on Cluster Computing and the Grid, 2007, pp. 541-548. CCGRID 2007 (2007)
29. Rountree, B., Lowenthal, D.K., Funk, S., Freeh, V.W., de Supinski, B.R., Schulz, M.: Bounding energy consumption in large-scale MPI programs. Proc. ACM/IEEE Conf. Supercomputing (2007)
30. Bunde, D.P.: Power-aware scheduling for makespan and flow. In: Proceedings of 18th Annual ACM Symposium Parallelism in Algorithms and Architectures (2006)
31. Kimura, H., Sato, M., Hotta, Y., Boku, T., Takahashi, D.: Empirical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster. In: IEEE International Conference on Cluster Computing, 2006, pp. 1–10. IEEE, NJ (2006)
32. Calheiros, R., Ranjan, R., Beloglazov, A., De Rose, C., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Experience **41**(1), 23–50 (2011)
33. VBoeres, C., Rebello, V.E.F.: A cluster-based strategy for scheduling task on heterogeneous processors. In: Proceedings of 16th Symposium on Computer Architecture and High Performance Computing, pp. 214–221. Foz do Iguacu (2004)