J. Parallel Distrib. Comput. 72 (2012) 1725–1740

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Energy-efficient deadline scheduling for heterogeneous systems

Yan Ma [a,b], Bin Gong [a,*], Ryo Sugihara [b], Rajesh Gupta [b]

[a] *School of Computer Science and Technology, Shandong University, Jinan Shandong 250101, China*

[b] *Department of Computer Science and Engineering, University of California, San Diego, CA 92093, USA*

ARTICLE INFO

Article history:
Received 19 August 2011
Received in revised form
10 May 2012
Accepted 28 July 2012
Available online 7 August 2012

Keywords:
Energy efficiency
Deadline scheduling
High performance computing
Pricing scheme

ABSTRACT

Energy efficiency is a major concern in modern high performance computing (HPC) systems and a power-aware scheduling approach is a promising way to achieve that. While there are a number of studies in power-aware scheduling by means of dynamic power management (DPM) and/or dynamic voltage and frequency scaling (DVFS) techniques, most of them only consider scheduling at a steady state. However, HPC applications like scientific visualization often need deadline constraints to guarantee timely completion. In this paper we present power-aware scheduling algorithms with deadline constraints for heterogeneous systems. We formulate the problem by extending the traditional multiprocessor scheduling and design approximation algorithms with analysis on the worst-case performance. We also present a pricing scheme for tasks in the way that the price of a task varies as its energy usage as well as largely depending on the tightness of its deadline. Last we extend the proposed algorithm to the control dependence graph and the online case which is more realistic. Through the extensive experiments, we demonstrate that the proposed algorithm achieves near-optimal energy efficiency, on average 16.4% better for synthetic workload and 12.9% better for realistic workload than the EDD (Earliest Due Date)-based algorithm; The extended online algorithm also outperforms the EDF (Earliest Deadline First)-based algorithm with an average up to 26% of energy saving and 22% of deadline satisfaction. It is experimentally shown as well that the pricing scheme provides a flexible trade-off between deadline tightness and price.

1. Introduction

With more and more applications relying on high performance computing (HPC), energy consumption is emerging as an important issue. Tianhe-1A topping the 36th Top500 list achieves a performance level of 2.57 petaflop/s, but also requires a power budget of 4.04 MW [64]. High energy consumption leads to low reliability of the system since the Arrhenius life-stress model shows that the failure rate of electronic devices will double as the temperature rises by every 10 °C (18 °F) [10]. Thus, efficient energy management is critical not only for green computing, but also for high reliability.

Generally, there are two ways to reduce power consumption in the HPC field. One is the low-power architectural approach that is implemented by changing the design of the architecture through built-in low-power components, such as Green Destiny [25] with Transmeta Crusoe processors, Blue Gene/P Solution [36] and NNSA/SC Blue Gene/Q Prototype [8] with the embedded PowerPC chip modified with floating point support. While this approach is effective, it results in expensive design and update overheads by relying on the specific components rather than common commodity parts. Another way called the power-aware software approach is more practical and regarded as a promising way [35]. The idea is to make a tradeoff between power and performance through power-aware algorithms. One such representative method is power-aware scheduling that considers energy consumption as one of the scheduling indicators and reduces energy consumption through power-saving techniques.

Dynamic Voltage and Frequency Scaling (DVFS) [66] and Dynamic Power Management (DPM) [6] are common system-level power saving techniques. DVFS provides a way of dynamically managing the power of processing elements by changing the supply voltages or operating frequencies. For example, the AMD Opteron processor has six levels of power consumptions ranging from 32 to 95 W, coming at the cost of operating frequencies from 1000 to 2600 MHz. Unfortunately, DVFS is applicable only to CPUs at the component level, which contributes only about one-third of the total system power [57,1]. Besides, some legacy CPUs do not support variable supply voltages or operating frequencies. In [57], Meisner et al. demonstrated that PowerNap, a DPM based solution they proposed, can provide greater energy efficiency than solutions based on DVFS. DPM techniques selectively turn off system components when they are idle and turn them on when they are requested, which is an efficient way to reduce the idle power. Hard disks, for example, consume an average of 6 W when

* Corresponding author.
E-mail addresses: yanpony@126.com (Y. Ma), gb@sdu.edu.cn (B. Gong), ryo@ucsd.edu (R. Sugihara), rgupta@ucsd.edu (R. Gupta).

0743-7315/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.07.006

they are idle [43], while processors even with AMD PowerNow! technology consume approximately 32 W when idle [4]. Present-day servers dissipate about 60% as much power when idle as when fully loaded [24,48]. Thus, power-aware scheduling should consider using the DPM approach based on no or less performance degradation or user-specified QoS requirements to reduce the idle energy cost.

Power-saving techniques aside, most previous studies address minimizing performance degradation when conserving energy [27,60,71,30,47], while we deal with power-aware scheduling to minimize QoS degradation in terms of meeting deadlines. In this paper, a novel power-aware deadline scheduling algorithm based on the DPM technique for heterogeneous systems is proposed. We focus on heterogeneous systems because they are more general than homogeneous systems, in which power-aware deadline scheduling is often straightforward through bin-packing type formulations. HPC platforms such as computational grids or data centers which house a number of machines together may cater for compute and resource intensive applications that take hours or days to execute, not only in contrast to the real-time applications that most studies on deadline scheduling for heterogeneous systems consider and rely on DVFS, but also different from the workloads made of small individual processing units such as http requests [53] that the resizing problem of computational pools addresses. As a type of application, we focus on bag-of-tasks (BoT) applications [14], i.e., parallel applications whose tasks are independent of each other. Despite their simplicity compared with workflows, a variety of problems in several fields, including computer imaging [26], computational biology [62], parameter sweeps [68], fractal calculations and data mining [15], have been modeled as bag-of-tasks applications. In the real world of high-performance (such as grids and clusters) and high-throughput computing, bag-of-tasks applications can easily execute on multiple resources to reduce the response time or meet the deadline constraints.

Our contributions in this paper are that we

- Formulate power-aware deadline scheduling for heterogeneous computing environments as an integer linear programming problem and also design a heuristic algorithm.
- Identify another subproblem related to task assignment on a single node and design an approximation algorithm.
- Propose a pricing scheme where the price of task execution is proportional to the total energy usage and largely depends on the deadline tightness.
- Make important extensions of the proposed algorithms for online scenarios and application scope, discuss the practical significance of the problem including the pricing scheme, and
- Demonstrate the effectiveness of the proposed algorithms by extensive experiments.

The rest of the paper is organized as follows. Section 2 introduces the related work. In Section 3 we formulate the master problem of power-aware deadline scheduling in heterogeneous systems. Section 4 discusses a slave problem of task assignment on a single node and an approximation algorithm is introduced. In Section 5 we design a heuristic algorithm to solve the master problem and give a novel pricing scheme. Section 6 extends the proposed algorithms to the online scenario and control dependence graph, and discuss the practical significance. Experimental results are presented in Sections 7 and 8 concludes the paper.

2. Related work

Our work belongs to aperiodic, synchronous, non-preemptive and independent task scheduling with deadline constraints. In aperiodic independent task scheduling, there are several classical algorithms. With respect to minimizing the maximum lateness, EDD (Earliest Due Date) is optimal for the synchronous scenario [40], EDF (Earliest Deadline First) is optimal for the asynchronous and preemptive scenario [19], and Tree Search is optimal for the asynchronous and non-preemptive scenario [7]. Refer to [61] for more details. There are also a number of studies on deadline scheduling in real-time or grid systems [34,20,22,65,42,54]. Their objectives are various: minimize the number of tardy jobs, optimize a fairness criterion, improve stability and robustness, minimize the makespan, consider the lowest cost, etc. However, they often do not consider the power or energy index.

Energy-efficient algorithms for deadline scheduling were first studied by Yao et al. [70], where they propose an offline algorithm, YDS, and two online algorithms, AVR and OA, that aim to minimize energy consumption for a set of independent jobs with deadline constraints on a single variable-speed processor. The authors have also extended their work to support discrete-voltage processors and tree-structured tasks [52,51]. There are several other studies on the single processor case. Irani et al. [38] extend the AVR algorithm to support the sleep state, first integrating the DVFS and DPM schemes. Chan et al. [11] propose the OAT algorithm, consider deadline scheduling on a processor with a fixed maximum speed and the system may be overloaded. Lately, Han et al. [33] investigate the sleep-aware version of the OAT algorithm called the SOA algorithm, which is 4-competitive for throughput and $(\alpha^\alpha + \alpha^2 4^\alpha + 2)$-competitive for energy (α is the constant involved in the speed-to-power function).

The listed efficient heuristics and theoretical analyses are developed for the single processor scenario. For multiprocessor systems, power-aware deadline scheduling can be categorized as per periodic or aperiodic task and homogeneous or heterogeneous systems. Most existing work investigates the homogeneous system [41,49,3]. Some recent work [32,13,12] considers the heterogeneous system, but they focus on periodic real-time tasks. In addition, all the above work mainly uses the DVFS scheme. Lately there is also some work on deadline scheduling considering temperature and thermal limits [69,58]. The background of our work is closely related to [58], which studies deadline scheduling for compute-intensive independent jobs without using DVFS in HPC data centers. One of the algorithms they propose (called SCINT) tries to minimize the energy consumption, but it is for preemptive tasks and also it is time-consuming due to the use of genetic algorithms.

In the context of HPC, there are several studies on energy-efficient deadline scheduling. Kim et al. [44] consider energy-optimization deadline scheduling for bag-of-tasks applications on homogeneous cluster systems. They propose a dynamic DVFS scheduling algorithm for both time-shared and space-shared resource sharing policies. Ma et al. [56] propose energy optimization scheduling for a DAG based application with multiple deadlines, combining DVFS and DPM through task clustering and binary search. Garg et al. [29] address environment-conscious deadline scheduling of HPC workloads on the distributed data centers. At the CPU level within a data center, they still use DVFS to reduce the number of deadline misses and energy consumption. These works are not applied to the heterogeneous system without supporting DVFS. Moreover, they just consider energy consumption from the viewpoint of providers, do not relate consumers' interest with energy or give consumers the flexibility of controlling energy.

Referring to bag-of-tasks applications on the HPC platform including grids and clusters, Cirne et al. [14] earlier developed MyGrid considering connectivity, security and heterogeneity issues to enable the user to run bag-of-tasks applications on grids. The input data and system scalability for scheduling are discussed in [17,67]. The authors in [2,5] respectively address the fault-tolerance and fair resource-sharing scheduling problem.

Scheduling works in [59,16,46,37] focus on the dynamics and heterogeneity of the execution environment while reducing the response time or makespan. With increased concerns on energy use, Kim et al. [44] consider power-aware scheduling of bag-of-tasks applications with deadline constraints on homogeneous clusters; To complete the similar work for heterogeneous systems, in this paper we formulate the problem and propose an effective algorithm.

Finally, there are some studies on applying a market-oriented model on power-aware scheduling. Subrata et al. [63] propose a cooperative, power-aware game theoretic solution to the job scheduling problem in grids. They directly take into account the multitude of ownerships of providers and maintain a specified response time. They focus on the transaction based jobs. A meta-scheduling model and several meta-scheduling policies are proposed in [29], aiming to minimize the carbon emissions and maximize the profit of the cloud provider. When computing the profit, the price users pay is given by the provider in units of CPU hours, which is similar to [9]. Li and Li [50]present utility based scheduling under constraints of deadline and energy budget. They try to maximize the utility of both parts and use resource price to interact. Resource price is determined by the previous resource allocation, and it is a tool of adjusting the utility, not directly related with energy consumption. At present there exists much work on pricing based scheduling strategies, as summarized in [55]. In short, most previous pricing schemes are based on demand and supply and usually are an iterative and slow process.

3. Master problem

In this section, we give the scheduling overview for the master problem of energy-efficient deadline scheduling in a heterogeneous multiprocessor environment and formulate it into an integer linear programming model.

3.1. Scheduling model

In traditional multiprocessor systems, the scheduling model consists of three layers: user layer, scheduling layer and resource layer. The user layer is in charge of submitting tasks. We focus on bag-of-tasks applications that can be divided into independent tasks, denoted as $\mathcal{T} = \{T_1 \cdots T_m\}$. As a scheduling unit, each task T_j has computation amount q_j (number of cycles) and deadline constraint d_j (min). The parameters are given by users when submitting. The scheduling layer accepts user requests and assigns tasks on resource nodes according to status information and scheduling strategy. Scheduling strategy is critical for saving energy and satisfying QoS requirements. The resource layer is responsible for task execution. The heterogeneous resource set is denoted as $\mathcal{P} = \{P_1 \cdots P_n\}$. Each node P_i has parameters with clock frequency and power consumption, denoted as f_i (Hz) and p_i (W), respectively. Heterogeneous systems in this study are abstract concepts, where parameters f and p are extracted. It is assumed that for an individual task, execution performance and energy efficiency depend on the above parameters and we do not go into details about the type of processing elements.

The task execution process in this scheduling model is as follows:

1. During a certain time window, bag-of-tasks applications are submitted by users. In high performance computing, waiting time is negligible because applications will execute for a long time ranging from several hours to several days. Since most data centers and supercomputing centers provision enough capacity to handle their peak utilization while the average usage is much lower [9], all or close to all requests can be accepted and scheduled. For the oversaturation case, we simply discuss it in Section 6.1.
2. The computation requests are sent to the scheduler and it makes a decision about node selection and task assignment according to the scheduling algorithm, requests information and system status including clock frequency, power consumption, real-time information and on/off mode.
3. Resource nodes begin to execute tasks. We focus on the non-preemptive task scheduling. Users pay for the usage of computation services, which depends on computation amount, pricing scheme and scheduling efficiency. About the execution time for a given task T_j running on a resource node P_i, we assume it can be estimated by q_j/f_i in order to facilitate the comparison of different policies [44]. In the model where users pay for usage, the formula also provides a fair and easy way of estimation, while overestimation/underestimation analysis and the influence of different parallel architectures is beyond the scope of this particular work.
4. Resource nodes return results to users. Suppose application arrival and task assignment to one resource node is intermittent, once the resource node finishes all the currently assigned tasks, it can shut down or sleep immediately and wait for the next assignment. Thus, resource nodes must first change from off (sleep) state to on state in the next assignment. This transition process needs an energy cost, denoted as e^{dpm}. The discussions here are for the static case. Refer to Section 6 for complex scenarios.

3.2. Problem formulation

The scheduling problem stated above is formally defined as follows. Given a heterogeneous node set $\mathcal{P} = \{P_i\}$ $(i = 1, \ldots, n)$ and an independent task set $\mathcal{T} = \{T_j\}$ $(j = 1, \ldots, m)$, where node P_i has power consumption p_i, clock frequency f_i and transition cost e_i^{dpm} and task T_j has size q_j and deadline d_j, assign tasks on nodes so that the total energy consumption is minimized while all the deadlines are satisfied. We call this problem *Energy Minimization Multitask Scheduling* (EMMS). Its decision version is defined by giving a constant K and asking whether there exists a task assignment with the total energy consumption no greater than K and all the deadlines satisfied.

For this problem, we have the following hardness result:

Theorem 1. *The decision version of EMMS is NP-complete.*

Proof. We consider a restricted case of homogeneous nodes ($\forall i.\ f_i = f, p_i = p, e_i^{dpm} = e$), identical deadlines ($\forall j.\ d_j = d$) and fixed number of nodes used l such as $l = n$. This case is equivalent to MULTIPROCESSOR SCHEDULING [28] due to the constant energy consumption. Since the decision version of EMMS is verified in polynomial time for a given assignment, it is NP-complete. □

Without loss of generality, we assume tasks are sorted in the non-descending order of deadline; i.e., $d_j \le d_{j'}$ for $j < j'$. Then the EMMS problem is formulated as

$$\min E = \sum_{i=1}^{n} \left\{ \frac{p_i}{f_i} \sum_{j=1}^{m} x_{ij} q_j + e_i^{dpm} y_i \right\} \tag{1}$$

$$\text{s.t. } y_i = \begin{cases} 1 & \sum_{j=1}^{m} x_{ij} \neq 0 \\ 0 & \sum_{j=1}^{m} x_{ij} = 0 \end{cases} \quad \forall i \tag{2}$$

$$\sum_{k=1}^{j} \frac{q_k}{f_i} x_{ik} \le d_j \quad \forall i, j \tag{3}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j \tag{4}$$

$$x_{ij} = \{0, 1\} \quad \forall i, j \tag{5}$$

where x_{ij} is the assignment variable that takes 1 when task T_j is assigned to node P_i, while y_i is the status variable that takes 1 when the resource node executes tasks. The linear inequality (3) is a constraint for satisfying the deadline of each task. This is based on the optimality of the Earliest Due Date algorithm [40].

Lemma 1. *The nonlinear constraint (2) can be replaced by an equivalent linear constraint:*

$$y_i \le \sum_{j=1}^{m} x_{ij} \le m y_i \quad \forall i,\ y_i = \{0, 1\}. \tag{6}$$

Proof. First validate inequalities (6) can be derived from Eq. (2). If $\sum_{j=1}^{m} x_{ij} = 0$ and $y_i = 0$, inequalities (6) hold; If $\sum_{j=1}^{m} x_{ij} \ne 0$, $y_i = 1$ and $1 \le \sum_{j=1}^{m} x_{ij} \le m$, inequalities (6) stand. Second prove Eq. (2) can be derived from inequalities (6). The apagoge approach is used. Assume inequalities (6) holds and Eq. (2) does not, which is equivalent to $\sum_{j=1}^{m} x_{ij} = 0$ and $y_i = 1$ or $\sum_{j=1}^{m} x_{ij} \ne 0$ and $y_i = 0$. If $\sum_{j=1}^{m} x_{ij} = 0$ and $y_i = 1$, $1 \le 0 \le m$ is obtained by substituting $\sum_{j=1}^{m} x_{ij} = 0$ and $y_i = 1$ in inequalities (6), which is false; If $\sum_{j=1}^{m} x_{ij} \ne 0$ and $y_i = 0$, $0 \le \sum_{j=1}^{m} x_{ij} \le 0$ is derived, which contradicts with the assumption $\sum_{j=1}^{m} x_{ij} \ne 0$. □

Based on the above analyses, we get the follow conclusion:

Theorem 2. *EMMS can be formulated into an Integer Linear Programming (ILP) model.*

4. Slave problem

Before giving the solution to the master problem, we first introduce and solve a slave problem as a bonus. In this section we formulate the slave problem, propose an approximation algorithm and give its performance analysis.

4.1. Problem statement

The slave problem is to choose a subset of tasks for bag-of-tasks applications and one resource node such that the total computation amount is maximized and the deadline constraints are satisfied, which is formally defined as follows. Task T_j has the fixed size q_j and a hard deadline d_j. Each task only executes on one node. A certain node has the fixed clock frequency f_0, and it can only execute one task at a time. The objective is to find the task group with the largest computation amount while satisfying all the deadlines. Its decision version is defined in a similar way as the one for the EMMS problem. We call this problem *Maximum Merging with Deadline Constraints* (MMDC).

By reduction from SUBSET-SUM using a restriction of $d_j = d$, $\forall j$, we can also show the NP-hardness of MMDC. Assuming the tasks are sorted in the non-descending order of deadline, the MMDC problem is formulated as an ILP as follows:

$$\max \sum_{i=1}^{m} x_i q_i \tag{7}$$

$$\text{s.t.} \sum_{i=1}^{j} \frac{q_i}{f_0} x_i \le d_j \quad \forall j \tag{8}$$

$$x_i \in \{0, 1\} \quad \forall i \tag{9}$$

where x_i is a selection variable that takes 1 when task T_i is in the subset.

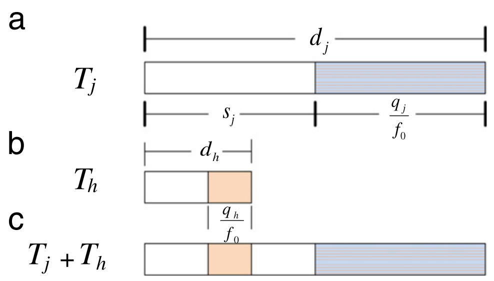

Fig. 1. Task placement and task merging.

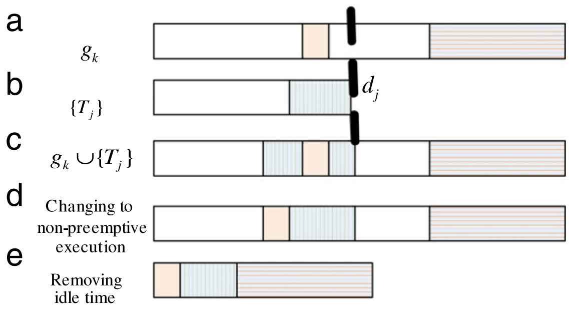

Fig. 2. Graphic representation of the initial task placement, the change from preemptive to non-preemptive execution and the removal of idle time.

4.2. Observation

For a task T_j, we define its execution time $t_j = q_j/f_0$ and introduce the idea of slack time $s_j = d_j - t_j$. Fig. 1(a) and (b) describe the case when executing tasks as late as possible. To fully utilize the resource, we then try to place one task into the slack time of another task. For instance, if the execution time of task T_h is less than the slack time of task T_j, $t_h < s_j$, and its deadline constraint is satisfied, task T_h can be "merged" with task T_j, as shown in Fig. 1(c).

4.3. Approximation algorithm

Based on the above observation, we propose an algorithm for the MMDC problem, called Largest-Slack-First Placement (LSFP). Given a task set and a resource node, the LSFP algorithm runs in an iterative way. Algorithm 1 shows the concrete steps. The scheduler first sorts tasks in the non-ascending order of slack time and selects the tasks that can be placed on the current node. In the algorithm, parameter $S_k(x)$ is introduced, denoting the sum of slack time in g_k before time point x, where g_k denotes the k-th consolidated task group. If $S_k(d_j) \ge q_j/f_0$ holds for the current task T_j, T_j is added to g_k. Our placement order is to put slack time s_j first and execution time t_j close to the deadline, just as Fig. 1 shows. The place order maintains the adjusting rule of $S_k(d_i)$ shown in Algorithm 1. Lastly, the algorithm selects and returns a task group $\tilde{g}$ with the largest computation amount.

The LSFP algorithm puts the current task close to its deadline as much as possible, which sometimes violates the non-preemptive execution rule as Fig. 2(c) shows. However, as shown in Fig. 2(d), it is always possible to swap the time slices so that the schedule becomes a non-preemptive one, without violating the deadline constraints. In the end, the idle time is removed as shown in Fig. 2(e).

Approximation analysis. Suppose OPT' is the optimal algorithm for MMDC, its output is the obtained largest computation amount and the corresponding task group, respectively denoted as Q^* and g^*. For the LSFP algorithm, $\tilde{Q}$ denotes the size of $\tilde{g}$. We have the following approximation result.

Algorithm 1 The LSFP algorithm

```
procedure LSFP($\mathcal{T}, f$)                    ▷ task set $\mathcal{T}$ and node speed $f$
    for $j = 1$ to $|\mathcal{T}|$ do
        $t_j = q_j/f$
        $s_j = d_j - t_j$
    $k \leftarrow 1$
    $\mathcal{T}' \leftarrow \mathcal{T}$
    while $\mathcal{T}' \neq \emptyset$ do
        $g_k \leftarrow \emptyset$
        $Q_k \leftarrow 0$
        Sort $\mathcal{T}'$ in the non-ascending order of $s_j$, i.e., $s_j \geq s_{j'}$ for $j < j'$
        $S_k(d_j) \leftarrow d_j,\ \forall j$ s.t. $T_j \in \mathcal{T}'$
        for $j$ s.t. $T_j \in \mathcal{T}'$ do
            if $S_k(d_j) \geq t_j$ then                        ▷ merging
                $g_k$=$g_k \cup \{T_j\}$
                $\mathcal{T}' = \mathcal{T}' \backslash \{T_j\}$
                $Q_k = Q_k + q_j$
                for $i$ s.t. $T_i \in \mathcal{T}'$ and $j < i$ do      ▷ adjusting $S_k(d_i)$
                    if $d_i \geq d_j$ then
                        $S_k(d_i) = S_k(d_i) - t_j$
                    else if $d_i > S_k(d_j) - t_j$ then
                        $S_k(d_i) = S_k(d_j) - t_j$
        $k \leftarrow k + 1$
    $\tilde{g} \leftarrow \arg\max_{g_k} Q_k$
    return $\tilde{g}$
```

Theorem 3. *$\tilde{Q} \geq Q^* - \min\{s_{\max}f, 2q_{\max} - q_{\min}\}$, where f is the frequency of the resource node, $s_{\max} = \max\{s_j\}$, $q_{\max} = \max\{q_j\}$, and $q_{\min} = \min\{q_j\}$.*

Please refer to Appendix B for the proof and Appendix A gives the preliminary knowledge.

5. Solution approach

In this section, we present a heuristic algorithm called EES to solve the master problem on the basis of the slave problem, discuss its approximation ratio and also propose a novel pricing scheme to relate users' interest with energy consumption.

5.1. Heuristic algorithm

The master problem is formulated into the ILP model in Section 3.2. We design a heuristic algorithm since solving the ILP problem is computationally expensive. There are three subproblems in EMMS: (i) selecting resource nodes; (ii) partitioning the task set into subsets; and (iii) mapping task subsets to resource nodes. We next give a heuristic algorithm based on the following metric.

For resource node P_i, we define energy cost in unit computation amount (unitcost for short) as follows:

$$c^i = \frac{E^i}{Q^i} = \frac{p_i t_i + e_i^{dpm}}{Q^i} = p_i/f_i + e_i^{dpm}/Q^i \tag{10}$$

where E^i (J) is the total energy consumption of P_i, Q^i (cycles) is the total finished computation amount, p_i is the power consumption of P_i, t_i is the time span for resource node P_i to execute Q^i, obtained by the ratio between Q^i and f_i, and e_i^{dpm} is the state transition cost. For simplification we assume $e_1^{dpm} = \cdots = e_n^{dpm}$.

In Eq. (10), an effective way to reduce the unitcost is to choose the resource node with small p_i/f_i. Based on this observation, we design a heuristic algorithm called Energy-Efficient Scheduling (EES). As shown in Algorithm 2, the EES algorithm works iteratively

Algorithm 2 The EES algorithm

```
procedure EES($\mathcal{T}$, $\mathcal{P}$)            ▷ task set $\mathcal{T}$, resource set $\mathcal{P}$
    while $\mathcal{T} \neq \emptyset$ do
        $P_0 \leftarrow \arg\min_{i:P_i \in \mathcal{P}} \{p_i/f_i\}$
        LSFP($\mathcal{T}, f_0$)
        Map $\tilde{g}$ on $P_0$
        $\mathcal{T} \leftarrow \mathcal{T} \backslash \tilde{g}$
        $\mathcal{P} \leftarrow \mathcal{P} \backslash \{P_0\}$
    Remove idle time for each task group
```

from the most power-efficient node. After choosing the node, by calling the LSFP algorithm to solve the MMDC problem discussed in Section 4, the EES algorithm assigns as many computation amounts as possible to that without violating the deadline constraints.

5.2. Theoretical analysis

In this section, we analyze the approximation result of the EES algorithm. The total energy consumption of the master problem consists of two parts: the energy consumed by executing tasks and the energy consumed by switching states of nodes, denoted as E_e and E_s, respectively. Suppose OPT is the optimal algorithm for EMMS, E_e^* and E_s^* denote its execution energy and switching energy.

Before the analysis, we first consider a special case. For a given task set $\mathcal{T}$ and node set $\mathcal{P}$, if all the tasks in $\mathcal{T}$ can be assigned to the most power-efficient node in $\mathcal{P}$ and all the deadlines can be satisfied at the same time, we call this case the *minimum optimal solution*. For the specific case, we have the following lemma.

Lemma 2. *For the case of the minimum optimal solution, the EES algorithm can find the optimal solution.*

Proof. In the proof, the method of mathematical induction is used. We assume the task set is denoted as $\mathcal{T} = \{T_j\}$, $j = 1, \ldots, m$ and the most power-efficient node is denoted as P_1. If the case is the *minimum optimal solution*, i.e., all the tasks can run on the node P_1 within their deadline constraints, we get that the inequalities

$$\sum_{k=1}^{j} q_k/f_1 \leq d_j, \quad \forall j \tag{11}$$

hold from the formulation in Section 3.2, where tasks are sorted in the non-descending order of deadline and denoted as $\{T_1, T_2, \ldots, T_m\}$. In the LSFP algorithm, we sort tasks in the non-ascending order of slack time and re-mark them as $\{T_1', T_2', \ldots, T_m'\}$.

Base case: if the case is the *minimum optimal solution*, the EES algorithm can put the first task T_1' on the node P_1 since it is inferred from the inequalities (11) that the execution time is not greater than its deadline for any task.

Inductive step: assume that the EES algorithm can put the first k, $2 \leq k < m$ tasks, $\{T_1' \cdots T_k'\}$, on the node P_1, i.e., during the process of the LSFP algorithm the parameter $S(d_j')$, $j = 1, \ldots, k$ of all the k tasks is not less than their respective execution time t_j', where $S(d_j')$ is the sum of slack time before time point d_j' and d_j' is the deadline of task T_j'. For the $(k+1)$-th task T_{k+1}', we next validate whether the EES algorithm can put it on the node P_1. Since the first k tasks can be put on the node, from the LSFP algorithm we know that after assigning one task, task T_{k+1}' needs to change its value of $S(d_{k+1}')$. Its initial value is d_{k+1}' and it needs to change k times. When the deadline of T_{k+1}' is not less than the deadline of T_j', $j = 1, \ldots, k$, the value of $S(d_{k+1}')$ is changed to $S(d_{k+1}') - t_j'$. When the deadline of T_{k+1}' is less than the deadline of T_j', $j = 1, \ldots, k$ and greater than $S(d'_j) - t'_j$, the value of $S(d'_{k+1})$ is changed to $S(d'_j) - t'_j$. As for the value of $S(d'_j)$, it is similarly computed by iteratively subtracting the execution time of the tasks whose deadline is less than d'_j. No matter the last change is $S(d'_{k+1}) - t'_k$, $S(d'_k) - t'_k$ or no change, we always obtain $S(d'_{k+1}) \geq t'_{k+1}$ because the inequalities (11) stand where the tasks are sorted in the non-descending order of deadline. Thus, the $(k+1)$-th task can be put on P_1.

Since the EES algorithm can put the first task on the node P_1, and the $(k+1)$-th task can also be put on P_1 on the assumption that the first k tasks can run on P_1, then the EES algorithm can put all m tasks on the most power-efficient node P_1 in the case of the *minimum optimal solution*. □

If the case is not the *minimum optimal solution*, we consider a more general case. The EES and OPT algorithms try to assign tasks among the node set $\mathcal{P} = \{P_i\},\ i = 1, \ldots, n$. Assume the nodes are sorted in the non-descending order of p_i/f_i; i.e., $p_i/f_i \leq p_{i'}/f_{i'}$ for $i < i'$. The EES algorithm first chooses the most power-efficient node, P_1, to assign tasks. It is easy to get that

$$p_1/f_1 \leq E_e^*/Q \tag{12}$$

follows, where Q is the total computation amount of task set $\mathcal{T}$. Suppose Q_1^* is the optimal solution to the MMDC problem for P_1 and $\mathcal{T}$, and Q_1 is the solution of the LSFP algorithm, we derive that

$$p_2/f_2 \leq E_e^*/(Q - Q_1^*) \tag{13}$$

is true since the second power-efficient node can cover the remaining computation amounts at the energy of at most E_e^* after P_1 finishes as many computation amounts as possible.

In Section 4.3, we get $Q_1 \geq Q_1^* - \min\{s_{\max}f, 2q_{\max} - q_{\min}\}$. Here we use z to denote $\min\{s_{\max}f, 2q_{\max} - q_{\min}\}$. If the EES algorithm can assign all the tasks on P_1 and P_2, we get the following lemma.

Lemma 3. *$E_e \leq (\frac{Q-X_0}{Q} + \frac{X_0}{X_0-z})E_e^*$, where $z = \min\{s_{\max}f, 2q_{\max} - q_{\min}\}$ and X_0 is the minimum computation amounts greater than z, which is a constant for a given $\mathcal{T}$ and P_1.*

Proof. If EES assigns all the tasks on P_1 and P_2, we get $E_e = \frac{p_1}{f_1}Q_1 + \frac{p_2}{f_2}Q_2 \leq (\frac{Q_1}{Q} + \frac{Q_2}{Q-Q_1^*})E_e^* \leq (\frac{Q_1}{Q} + \frac{Q-Q_1}{Q-Q_1-z})E_e^*$, where Q_2 is the computation amounts assigned to P_2 and $Q_2 = Q - Q_1$. Next we analyze the function $f(Q_1) = \frac{Q_1}{Q} + \frac{Q-Q_1}{Q-Q_1-z}$. The range of variable Q_1 is $[0, Q-z)$ since $0 \leq Q_1 < Q$ and $0 < Q_1 + z < Q$ stand and it is not the case of the *minimum optimal solution*. The derivative of $f(Q_1)$ is $f'(Q_1) = 1/Q + z/(Q - Q_1 - z)^2$. Since $f'(Q_1) > 0$, we know that $f(Q_1)$ is an increasing function on the range of $[0, Q-z)$. Each task can only run on one node, thus we can get all the combinations of computation amount, $\mathcal{Q} = \{q_1, q_2, \ldots, q_m, q_1+q_2, q_1+q_3, q_2+q_3, \ldots, q_1+q_2+\cdots+q_m\}$, using $q_1, \ldots, q_m$. Assume X_0 is the minimum computation amount greater than z in $\mathcal{Q}$ and for a given $\mathcal{T}$ and $\mathcal{P}$ we can easily obtain the value of X_0. Therefore, $E_e \leq (\frac{Q-X_0}{Q} + \frac{X_0}{X_0-z})E_e^*$ is derived. □

Suppose Q_2^* is the largest assigned computation amount after trying P_1 and P_2 using some optimal solution OPT_2, we get that

$$p_3/f_3 \leq E_e^*/(Q - Q_2^*) \tag{14}$$

stands because the third power-efficient node can cover the remaining computation amounts at the energy of at most E_e^* after P_1 and P_2 finish as many computation amounts as possible.

Similarly, let Q_i^* denote the largest assigned computation amounts after trying the first i nodes using OPT_2 and we derive that

$$p_{i+1}/f_{i+1} \leq E_e^*/(Q - Q_i^*), \quad i = 1, 2, \ldots, n-1 \tag{15}$$

follows. For the EES algorithm, we get

$$\begin{aligned} E_e &= \frac{p_1}{f_1}Q_1 + \frac{p_2}{f_2}Q_2 + \cdots + \frac{p_n}{f_n}Q_n \\ &\leq \left(\frac{Q_1}{Q} + \frac{Q_2}{Q - Q_1^*} + \cdots + \frac{Q_n}{Q - Q_{n-1}^*}\right)E_e^* \end{aligned} \tag{16}$$

where Q_i is the largest computation amounts assigned to P_i using the EES algorithm, $\sum_{i=1}^n Q_i = Q,\ Q_i \geq 0$.

The EES algorithm may end in the k-th iteration since $Q_1 + Q_2 + \cdots + Q_k = Q,\ k \in [1, n]$. In this case $Q_k^* = Q$ must hold because the EES algorithm is worse than some optimal algorithm OPT_2 which tries to assign as many computation amounts as possible using k nodes. Meanwhile, for the OPT_2 algorithm it may also end when $Q_j^* = Q$. From the idea of the EES and OPT_2 algorithms, we get $j \leq k$. Lemmas 2 and 3 respectively discuss the case when $k = 1$ and $k = 2$. Properly speaking, the two cases are $k = 1, j = 1$ and $k = 2, j = 2$ since the value of j cannot be 1 when $k = 2$ by analyzing Lemma 2. For the other cases, i.e., $3 \leq k \leq n,\ 1 < j \leq k$, we have the following lemma.

Lemma 4. *$E_e \leq \max\{\frac{Q}{q_{\min}}, \frac{p_k/f_k}{p_1/f_1}\}E_e^*$, where k is the number of iterations in the EES algorithm.*

Proof. Based on the above analysis, we get $E_e \leq (\frac{Q_1}{Q} + \cdots + \frac{Q_j}{Q-Q_{j-1}^*})E_e^* + \frac{p_{j+1}}{f_{j+1}}Q_{j+1} + \cdots + \frac{p_k}{f_k}Q_k,\ 3 \leq k \leq n,\ 1 < j \leq k$. If $(j+1) > n$ or $(j+1) > k$, $\frac{p_{j+1}}{f_{j+1}}$ is assigned to 0. Since $Q \geq Q - Q_1^* \geq \cdots \geq Q - Q_{j-1}^*$ and $\frac{p_1}{f_1} \leq \frac{p_2}{f_2} \leq \cdots \leq \frac{p_k}{f_k}$, we get $E_e \leq \frac{\sum_{i=1}^j Q_i}{Q-Q_{j-1}^*}E_e^* + \frac{p_k}{f_k}\sum_{i=j+1}^k Q_i$. Since each task can only execute on one node, i.e., no migration, the minimum value of $(Q - Q_{j-1}^*)$ is $q_{\min}$ no matter what the value of j is. Then $E_e \leq \frac{\sum_{i=1}^j Q_i}{q_{\min}}E_e^* + \frac{p_k}{f_k}\sum_{i=j+1}^k Q_i$ is true, i.e., $\frac{E_e}{E_e^*} \leq \frac{\sum_{i=1}^j Q_i}{q_{\min}} + \frac{\frac{p_k}{f_k}\sum_{i=j+1}^k Q_i}{E_e^*}$. From inequality (12), we know $E_e^* \geq \frac{p_1}{f_1}Q$ follows and the ratio is changed to $\frac{E_e}{E_e^*} \leq \frac{\sum_{i=1}^j Q_i}{q_{\min}} + \frac{\frac{p_k}{f_k}\sum_{i=j+1}^k Q_i}{\frac{p_1}{f_1}Q}$. Since $Q_1 + \cdots + Q_k = Q$, $\frac{E_e}{E_e^*} \leq \max\{\frac{1}{q_{\min}}, \frac{p_k/f_k}{Qp_1/f_1}\}Q = \max\{\frac{Q}{q_{\min}}, \frac{p_k/f_k}{p_1/f_1}\}$ is derived. □

Lemmas 2–4 discuss the approximation ratio of execution energy in different cases of assignment. For the case of the *minimum optimal solution*, $E_e = E_e^* \leq \max\{\frac{Q}{q_{\min}}, \frac{p_k/f_k}{p_1/f_1}\}E_e^*$ holds. For the case of $k = j = 2$, we get $E_e \leq \frac{Q}{q_{\min}}E_e^*$ inspired by Lemma 4. Combining $E_e \leq \frac{Q}{q_{\min}}E_e^*$ and Lemma 3, we derive $E_e \leq \min\{\frac{Q-X_0}{Q} + \frac{X_0}{X_0-z}, \frac{Q}{q_{\min}}\}E_e^* \leq \max\{\frac{Q}{q_{\min}}, \frac{p_k/f_k}{p_1/f_1}\}E_e^*$ for the case of $k = j = 2$. Therefore, for all the cases we conclude that the approximation ratio between E_e and E_e^* is

$$E_e \leq \max\left\{\frac{Q}{q_{\min}}, \frac{p_k/f_k}{p_1/f_1}\right\}E_e^*. \tag{17}$$

If the state switching energy of a node is zero, inequality (17) gives us the approximation ratio between the EES algorithm and OPT algorithm; else please see Theorem 4.

Theorem 4. *EES $\leq \max\{n, \frac{Q}{q_{\min}}, \frac{p_k/f_k}{p_1/f_1}\}$OPT, where k is the number of iterations in the EES algorithm.*

Proof. The total energy consumption consists of execution energy and switching energy for the EMMS problem. For the switching energy, we get $E_s \leq nE_s^*$ assuming different nodes have the same switching energy, i.e., $e_1^{dpm} = \cdots = e_n^{dpm}$; for the execution energy, we derive $E_e \leq \max\{\frac{Q}{q_{\min}}, \frac{p_k/f_k}{p_1/f_1}\}E_e^*$, where k is the number of iterations in EES. Thus, $E_s + E_e \leq nE_s^* + \max\{\frac{Q}{q_{\min}}, \frac{p_k/f_k}{p_1/f_1}\}E_e^* \leq \max\{n, \frac{Q}{q_{\min}}, \frac{p_k/f_k}{p_1/f_1}\}(E_s^* + E_e^*)$, i.e., $EES \leq \max\{n, \frac{Q}{q_{\min}}, \frac{p_k/f_k}{p_1/f_1}\}$ OPT is proved. □

Note that $E_s \leq nE_s^*$ is a very loose upper bound. In fact, the ratio $\frac{E_s}{E_s^*}$ is much less than n during the assignment. For the execution energy, the worst-case upper bound of $\frac{E_e}{E_e^*}$ is $\max\{\frac{Q}{q_{\min}}, \frac{p_n/f_n}{p_1/f_1}\}$. As for the approximation ratio between EES and OPT, although it seems to be large, the performance of the EES algorithm is close to the optimal algorithm as evaluated in Section 7.

5.3. Task pricing scheme

The idea of unitcost improves the energy efficiency of scheduling in a heterogeneous environment, but it can also be useful for a market-oriented scenario, where users pay a monetary cost for submitting their tasks. The price of task execution depends on the cost of energy it takes, and that is affected by its deadline. By relating energy cost and monetary cost in this way, we can further promote energy-efficient computing not only at the provider's side but also at the users' side.

Based on unitcost, we define the monetary cost C of task T_j executed at resource node P_i as follows:

$$C = \alpha v q_j c^i, \tag{18}$$

where α is the coefficient of profit margin, v is the price of unit electricity, q_j is the size of task T_j, and c^i is the unitcost of resource node P_i under a given task group. Note that C is proportional to the actual energy cost. In the formula, α is a constant providers control, e.g., a provider can set its profit margin to 20%; v is decided by the electric power companies or the government, e.g., the price of commercial electricity in Beijing, China in 2011 was set to 0.794 RMB/kWh [23]; q_j is the computation amount of the j-th task and is given by the user when submitting; c^i depends on the task assignment and is computed as Eq. (10), where i is the index of the node the task is assigned to.

The assignment is influenced by the deadline setting of different users. Through the proposed algorithms, the users know that if the deadline of a task is tight they may have to pay more money because the task is likely to be assigned to the less power-efficient node and the other tasks should accommodate it. The users thus may combine the rule and their specific performance requirements to make decisions about the deadline. In this case, it is better for users to prolong their deadlines within acceptable limits when they don't know the information of each before submission. Based on the input we get the assignment through the EES algorithm and also relate the monetary cost with energy consumption through the proposed pricing scheme. We demonstrate the relationship between cost reduction and deadline increase in Section 7.3 to sustain the idea. Moreover, the EES algorithm tries to fully utilize the power-efficient resource node to execute tasks. In this way, the resource nodes with the lower power efficiency will be eliminated in the end because they cannot obtain tasks to execute. Thus, the EES algorithm can guarantee the energy-oriented survival of the fittest.

Adjustment phase. We just now discussed the situation when there is no adjustment phase during the submission and execution phases. Users can increase their deadline to obtain the large possibility of cost reduction. If the adjustment phase is added, users are given another chance to choose the deadline and certainly reduce cost. The objective of the design is to further reduce the total energy consumption. In order to improve the motivation of users in energy conservation, we give the users who increase their deadline economic interests, while the tasks not changing their deadline are unaffected. The deadline is not allowed to decrease in the adjustment phase. The adjustment works as follows.

The tasks increasing their deadline are sorted according to the non-ascending order of power efficiency of the nodes they are initially assigned to. If two tasks are assigned to the same node, the task with the larger deadline/size ratio is given a higher priority, where the deadline here is the first given deadline. This means it can also raise the priority in the adjustment phase for a task initially choosing a large deadline, besides the potential cost reduction. In this order, each task increasing deadline is reassigned to the node with higher power efficiency than that of the node it is first assigned to according to the currently fixed assignment and its new deadline constraint. The most power-efficient one of all the nodes where its new deadline can be satisfied is selected. Since the total energy consumption reduces as long as the tasks move to the more power-efficient node, providers ignore the effects of state transition cost of the node. The cost of the tasks not changing their deadline stays the same, while the part influenced by the power efficiency, p/f, in the cost of the tasks increasing their deadline is substituted by the value of the new node. In this way, users can control their deadline to make more informed decisions in the adjustment phase.

6. Extension and discussion

So far, we have formulated a power-aware deadline scheduling problem for heterogeneous systems, proposed its static heuristic algorithm and dealt with a task pricing scheme based on unitcost metric. In this section, we analyze its online scenario, extend its application fields and discuss the practical significance.

6.1. Online scenario

In Sections 4 and 5, we assume that the algorithm takes decisions based on knowledge of the details of the entire task set. In the realistic scenarios, tasks are submitted and scheduled dynamically. We now extend this to deal with the online and complex cases.

In order to make a better mapping and adapt to the online scheduling, we introduce the notion of scheduling cycle, i.e., at a fixed time interval the scheduler assigns tasks to the heterogeneous system. We have discussed the algorithm in one scheduling cycle. When the older tasks are being processed and a new scheduling cycle starts, the EES and LSFP algorithms can also be easily extended and changed to address the dynamics and continuity. However, there are several issues as follows:

Information collection. At the beginning of a scheduling cycle, the scheduler needs to collect the information of the newly arrived tasks, the old unfinished tasks and the status of resource nodes. The scheduler also maintains a sequence of the resource nodes and their current unfinished tasks in the non-ascending order of power efficiency. A given resource node, for example P_i, currently has an assigned but undone task set, g^i. The newly arrived task set is denoted as $\mathcal{T}_{\text{new}}$.

Time horizon. The scheduler records the current scheduling time t_{cur} in the time horizon. When a task arrives during a scheduling cycle, the user gives its relative deadline $d_j^{r'}$ and then the absolute deadline is computed as $d_j = d_j^{r'} + t_{\text{cur}}$, where t_{cur} is the scheduling time for the arrival cycle. For the unfinished or unexecuted tasks arrived at the previous cycles, we view them as the ones arrived in the current scheduling cycle, where their absolute deadlines remain the same and the sizes are updated using the remaining computation amount.

Parameter computation. For each resource node, say P_0, the important parameter $S_0(d_j)$, introduced in Section 4.3, of its old unfinished tasks $g^0 = \{T_j\}, \ \forall j$ need to be precomputed before a new cycle, where $S_0(d_j)$ denotes the sum of slack time on processor P_0 before the time d_j. Its computation method follows the same adjusting rule in Algorithm 1. At first, its initial value is given by the new relative deadline $d_j^r = d_j - t_{\text{cur}}$, where t_{cur} is the scheduling time of the new cycle. Then the value is updated in the non-ascending order of the initial slack time, which is obtained by $s_j = d_j - t_j - t_{\text{cur}}$ for task T_j, where t_j is the updated execution time using the remaining computation amount.

The selection of resource node. Due to the unitcost metric, similar to the static case, we select from the most power-efficient node for the online scenario. Differently, the initial status of a node P_0 during the scheduling cycle may be on or off. If the node P_0 is off, we directly use Algorithm 1 to find its assigned task group; if the node P_0 is on, the online algorithm still follows the framework of the LSFP algorithm and just needs to update several parameters such as the initial task group, the sum of slack time before a time point, etc. Please refer to Algorithm 4 in Appendix C for more details.

From Appendix C, we know that the online algorithm appends each newly arrived task set to the existing schedule and any request that is already in the schedule will never be cancelled to accommodate later arriving requests, which reduces the execution cost of the scheduling algorithm and migration cost of the application. The online scenario is in fact the version of the static algorithm that starts executing from the middle of the assignment.

Admission control. In a scheduling cycle, whenever requests arrive, the scheduler can decide whether to admit or reject some of the requests. There are two cases for a scheduler to reject a request. Users give the relative deadline when submitting a task T_j. If the deadline is too tight, i.e. $d_j^{r'} < q_j/f_{\max}$, where $f_{\max}$ is the maximum speed for the heterogeneous system, no node can finish it within the deadline and the scheduler will reject this kind of request. The premise is that we use q/f to estimate the execution time as discussed before. The other scenario of rejection is oversaturation by requests. In this case, the scheduler will invoke the scheduling algorithm before the new cycle and then decides whether the task can be included or omitted using the metric of deadline satisfaction.

6.2. *Control dependence graph*

We focus on bag-of-tasks applications in the previous part. In order to enlarge the application scope of our algorithm, we use a novel decomposition approach to view task graphs as the phased bag-of-tasks applications. Precedence-constrained applications usually use a directed acyclic graph (DAG) model, where cyclic dependencies among tasks in some cases can be eliminated [18] through branch prediction, parallel loop unrolling or sequential loop elimination, etc. methods.

In the user layer, if users submit precedence-constrained applications, we assume the parameters of each task (including q and d) and dependency of the tasks are given ahead of time and the scheduler assigns tasks to the heterogeneous system in the form of a scheduling cycle. In the time horizon, the current scheduling time is recorded in t_{cur}, the initial time of all the tasks in the application arrived during the current scheduling cycle is equal to t_{cur}, and the absolute deadline for each task is naturally computed as $d = d^{r'} + t_{\text{cur}}$, where $d^{r'}$ is the given relative deadline.

Differently, in each scheduling cycle we dynamically record a set of currently active tasks for each application. The active tasks refer to the unfinished tasks with no predecessors or where all the predecessors are finished, and thus all the active tasks for the arrived applications constitute a set of independent tasks. The active tasks consist of two parts: assigned to some processing element (say P_i) but undone task set g^i and the newly arrived task set $\mathcal{T}_{\text{new}}$. When an application arrives, its entry tasks[1] are first added into $\mathcal{T}_{\text{new}}$. Once all the predecessors for a task T_j are finished, task T_j is also added into $\mathcal{T}_{\text{new}}$. As for the other issues, it is the same with that in Section 6.1.

In the extension we focus on the applications with control dependence, where communication cost is negligible. This is how the EES algorithm is designed. If communication effects are taken into account, the dynamics of communication cost with the change of assignment makes the scheduling complicated and we need to make substantial changes to the original algorithm. Although computation and communication operations can be assumed to proceed simultaneously for data dependence graphs, we leave this part, especially for communication-intensive applications, to another separate work.

6.3. *Practical significance*

In this paper, we address the energy-efficient deadline scheduling for heterogeneous systems. For the application, we focus on bag-of-tasks applications running on the HPC platform such as computational grids, which can apply to various scenarios [15], such as Monte Carlo simulations, massive searches (e.g. key breaking), parameter sweeps, image manipulation, and data mining. We then extend it to control dependence graphs, which further extends its application.

For the system, we address the heterogeneous systems, which have the good performance for the compute intensive applications. With the development of technology and the on-going upgrading of the platform, heterogeneous systems have become more common.

For the scheduling indicator, we consider energy consumption and deadline constraints. Energy-aware/power-aware scheduling algorithms are drawing attention with the increased importance of energy management in the HPC field, as shown in [4,63]. For the deadline constraints, most schedulers can give the delay bound or deadline parameter to let users specify their requirements. Many real world examples in high performance computing, especially for the model of utility computing, have deadline limits, e.g. see [45].

We also discuss a market model based on the proposed algorithm. A user submits its requests to a given heterogeneous system and the user needs to pay money for the computational service according to its energy consumption, which depends on the task assignment. In this case, the user cannot know the price in advance. However, we can address the problem in several ways. Before the actual submission, the scheduler can perform pre-assignment to determine the price for the task execution. For the tasks with special requirements, the scheduler could also provide the parameter of specifying the mapped node, which will not affect our algorithms and where we just need to add these tasks to the old unfinished task set of the specified node ahead of time. Even if no steps are adopted, the lower bound and upper bound of the cost can be obtained. Using $C(T_j)$ to denote the cost task T_j pays to the system $\mathcal{P}$, its range is

$$C(T_j) \in [\alpha v q_j (p/f)_{\min}, \ \alpha v q_j (p/f)_{\max} + \alpha v e^{dpm}] \tag{19}$$

where $(p/f)_{\min} = \min\{p_i/f_i\}, \ \forall i, P_i \in \mathcal{P}$ and $(p/f)_{\max} = \max\{p_i/f_i\}, \forall i, P_i \in \mathcal{P}$, representing the unit power of the most and least power-efficient node.

7. Experimental results

In this section we evaluate the performance of the static and online version of the EES algorithm in the context of synthetic workload and workload generated from realistic data.

[1] A task without any predecessor is called an entry task.

Table 1
Parameter of heterogeneous nodes.

Server	#	f (GHz)	p (W)
IBM	1	2.13	675
	2	2.13	670
	3	2.66	1440
	4	2	1350
	5	1.86	1975
	6	2.33	310
	7	2.26	670
	8	2	835
	9	2	675
	10	3	400
HP	11	2	460
	12	2	750
	13	2.4	460
	14	2.4	300
	15	2.4	920
DELL	16	2.4	345
	17	2.4	305
	18	2.13	345
	19	2.26	1100
	20	2.33	345

7.1. Methods

The proper and representative benchmarks are important. For static EES, we compare it with the OPT, LP and EDD algorithms. OPT is the optimal solution to EMMS obtained by solving the ILP problem presented in Section 3.2. Since OPT is hard to solve when the scale is large, we introduce LP, which is obtained by relaxing the integer constraint (Inequality (5)) of the ILP formulation. We use lp_solve 5.5.2.0 package for the LP solver. EDD works for independent tasks with the same arrival time, which is a simple form of EDF [19]. For online EES, we compare it with the EDF algorithm. EDF is a classic deadline scheduling algorithm for independent tasks with arbitrary arrival times and first executes the task with the earliest absolute deadline. To adapt to heterogeneous systems, the EDD and EDF algorithms first sort the resources in the non-ascending order of power efficiency.

For the configuration of heterogeneous systems, we use specification parameters from 20 real-world servers sold by IBM, HP and Dell, as shown in Table 1. Since it takes 200 W on average and one to two minutes to boot up a node, we set $e^{dpm} = 20$ (kJ).

Synthetic workload can thoroughly and quickly validate the algorithm using various input parameters with low cost. Table 2 shows the rules we use for generating the synthetic task sets. All random parameters obey the uniform distribution. Three node sets $\mathcal{P}_i$ $(i = 1, 2, 3)$ are made using the above servers and assigned for each of small/medium/large-scale workload. The unit of task size is teracycle, which embodies the long execution time. The deadline of a task is set to be greater than the task size divided by the slowest frequency (1.86 GHz) so that it is always no less than the execution time at any node. In other words, any task can be executed at any node without violating the deadline constraint if we ignore all other tasks in the task set. Note that this does not guarantee the existence of a feasible schedule for the task set. Idle time is set by 20%–50% of the average execution time for the given task size range.

Realistic workload helps verify the feasibility of the solution in practical use. We use DAS-2 trace from Grid Workload Archive (GWA) [31] and ANL Intrepid trace from Parallel Workload Archive (PWA) [21]. DAS-2 traces are published by Advanced School for Computing and Imaging (the owner of the DAS-2 system), which provide two job structures: unitary and BoT. ANL Intrepid traces are from Intrepid system deployed at Argonne Leadership Computing Facility (ALCF) of Argonne National Laboratory and

Table 2
Generation rule of synthetic workload.

Workload parameter	Small-scale	Medium-scale	Large-scale
Number of tasks	[1, 25]	[26, 80]	[81, 100]
Resource nodes	$\mathcal{P}_1 = \{1\text{–}20\}$	$\mathcal{P}_2 = 2\mathcal{P}_1 \cup \{1\text{–}10\}$	$\mathcal{P}_3 = 3\mathcal{P}_1$
Task size (teracycle)	[20, 200]		
Task deadline (10^3 s)	[task size/1.86 GHz, task size/1.86 GHz+ idle time]		
Idle time (10^3 s)	[12, 30]		

Table 3
LP-normalized energy consumption.

	EDD		EES		OPT[a]	
	Mean	s.d.	Mean	s.d.	Mean	s.d.
Small-scale	1.028	0.011	1.008	0.003	1.006	0.002
Medium-scale	1.143	0.112	1.041	0.033	–	–
Large-scale	1.256	0.222	1.047	0.057	–	–

[a] "–" means that the optimal solution cannot be found within a reasonable time.

used primarily for scientific and engineering computing. We obtain the parameters of submit time and runtime from the log. In order to fit system heterogeneity and simplify computation, we compute the size of the task according to the runtime and its execution speed [44]. Since our scheduling unit is task instead of job, we choose the longest one of the tasks for each job when the job is assigned to multiple processors. As for the deadline, we use the method in [39] to assign deadlines through the ratio between deadline and runtime. If the ratio of deadline/runtime for a task is small, e.g. between 2 and 6, we say the task has a tight deadline. If the ratio is large, e.g. between 10 and 14, the task has a loose deadline. About the scheduling cycle, we use 10 min as the time interval to not only maintain the low scheduling cost but also reduce the unnecessary large delay.

7.2. Static EES

We now analyze the effectiveness of the static EES algorithm when it respectively runs on synthetic workload and realistic workload from GWA.

7.2.1. Synthetic workload

In this section we validate the energy reduction capacity with the change of various parameters such as deadline, utilization rate and the number of tasks, and also consider the performance including deadline violation rate and energy optimization when the deadlines are tight.

To calculate the average energy consumption, we generate 100 instances for each of the different scale synthetic workloads. For the convenience of comparison, the average energy consumption of each algorithms is normalized by the solution of LP. Table 3 shows the comparative results of the EDD, EES and OPT algorithms. For the small-scale workloads, EES uses 1.98% less energy than EDD does, and is only 0.20% more than OPT. For medium and large systems, EES is within 5% worse than LP and 9.8% (medium) and 19.96% (large) better than EDD. The standard deviation values show that EES also has less variation in performance, especially for medium-scale and large-scale workloads. Running time of EES is less than 0.2 s for all cases.

The number of tasks. Fig. 3(a) explores how the number of tasks affects energy consumption. Since OPT cannot be solved in a realistic time for the large problems, its curve is incomplete. With the increase of the number of tasks, the advantage of EES over EDD is more obvious while the relative difference between LP and EES is smaller.

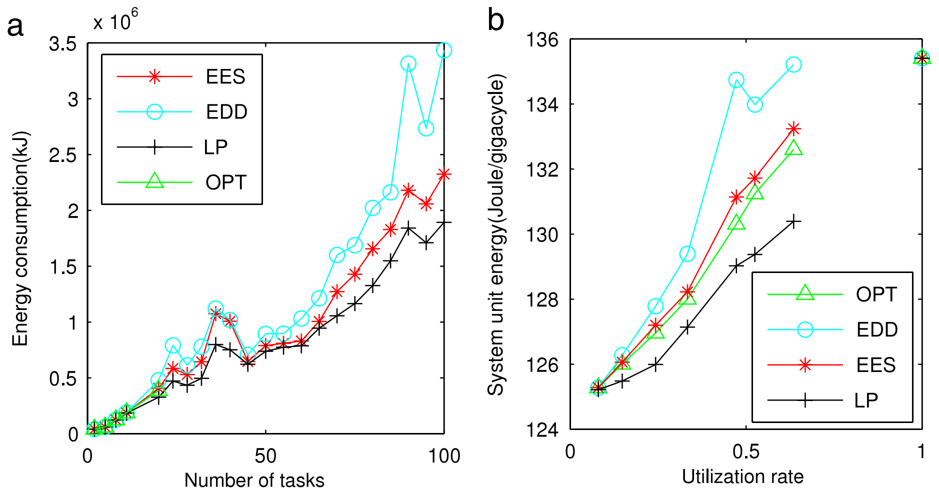

Fig. 3. (a) Energy consumption in different numbers of tasks. (b) Average system unit energy in different utilization.

Utilization rate. Next we analyze the effect of "utilization", i.e., how much of the idle time of all nodes is filled by tasks. We use the following definition of utilization:

$$u = \frac{\sum_{T_j \in \mathcal{T}} q_j}{\sum_{P_i \in \mathcal{P}} f_i \max\{d_j\}} \tag{20}$$

primarily because this value is unique to the task set and is insensitive to how the tasks are assigned to the nodes, i.e., insensitive to the choice of scheduling algorithms. Fig. 3(b) shows the relationship between utilization and "system unit energy", which is defined as $c = E/Q$, where E is the total energy $\mathcal{T}$ consumed and Q is the total size of $\mathcal{T}$. EES has a good energy-utilization efficiency close to OPT while EDD is much worse than EES. The energy consumption for the case when the utilization rate equals 1.0 is a theoretical value calculated by assuming all nodes are active and there is no slack time. We do not have enough data points for high utilization rate close to 1.0, since it is generally hard to obtain a randomly-generated task set that is feasible.

Deadline tightness. Next we study how energy consumption is affected when the deadline is tight or even violated. Fig. 4 shows the comparative performance in tight deadline. For this case, we set the deadline of task T_j as $q_j/2.13$. Although the energy saving ability of all algorithms become worse than the case with the generation rule in Table 2, EES still has 4.92%–43.58% more advantage over EDD.

For the case when some deadlines are violated, the EES algorithm tries to not only save energy consumption, but also find a solution satisfying all the deadlines. Next we explore its deadline satisfaction ability. We respectively run 10 times for 100 random instances in different scales. Table 4 gives the results of average deadline violation in different deadlines. We set deadline separately as $d = 2q/\max(f)$, $d = q/\min(f)$ and $d = q/\text{mean}(f)$, which can show different situations of deadline violation. For example, when $d = q/\text{mean}(f)$, both EES and EDD cannot find a feasible solution for the large-scale workload, which shows that this deadline is too tight for the large-scale instances; when $d = 2q/\max(f)$, all the algorithms do not violate deadlines for the small-scale workload, which shows that this deadline is not tight for the small-scale instances. Excluding the same value in the table, we obtain that EES is close to OPT and 19.3% better than EDD in deadline satisfaction.

Deadline. We have analyzed the results for both loose and tight deadlines. Next, Fig. 5 shows how energy consumption changes

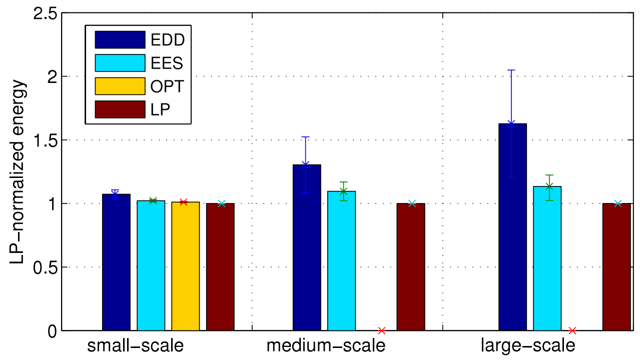

Fig. 4. Mean and standard deviation value of LP-normalized energy in tight deadline.

with the increase of deadline for the same instance in small-scale and large-scale systems. The energy saving ability of EES increases monotonically with the increasing deadline, while EDD sometimes has exception and fluctuation as shown in Fig. 5(a). Energy consumption of EES will eventually approach to that of OPT or LP, while energy in EDD right now reaches a certain fixed value higher than EES. Moreover, the number of points for different algorithms is different. This is because EDD, EES, and OPT cannot find feasible solutions when the deadline is tight enough.

7.2.2. Realistic workload

Until now we give the comprehensive and statistical validation of static EES using synthetic instances. In order to further evaluate its practicality and efficiency, we now run the static EES algorithm on the DAS-2 workload traces from GWA.

We select the workloads in the first 20 time intervals of 10 consecutive minutes since 10 min is assumed to be a scheduling cycle. For each 10 min workload, every task in the workload is assigned a tight or loose deadline in the way Section 7.1 describes. The range of deadline/runtime ratio for a loose deadline is set from 10 to 14, while that for a tight deadline is from 2 to 6. The percentage of tight deadlines gradually changes and we get the average energy consumption. According to the scale of workload we set the system composed of $10\mathcal{P}_1$. OPT and EDD may not have feasible solutions for some instance and we eliminate the results of the instance during the computation.

Table 5 shows the comparative results for the different percentage of tight deadlines. EDD performs the worst, especially when the percentage of tight deadlines is equal to 100%. EDD is

Table 4
Percentage of average deadline violation in different deadlines.

	$d = 2q/\max(f)$				$d = q/\min(f)$				$d = q/\text{mean}(f)$			
	EES (%)	EDD (%)	OPT (%)	LP (%)	EES (%)	EDD (%)	OPT (%)	LP (%)	EES (%)	EDD (%)	OPT (%)	LP (%)
Small-scale	0	0	0	0	6.4	7.6	6.2	0	52.9	57.1	51.6	0
Medium-scale	0	2.8	–	0	23.2	35.4	–	0	92	96.9	–	0
Large-scale	0.1	14.7	–	0	87.6	98.3	–	0	100	100	–	0

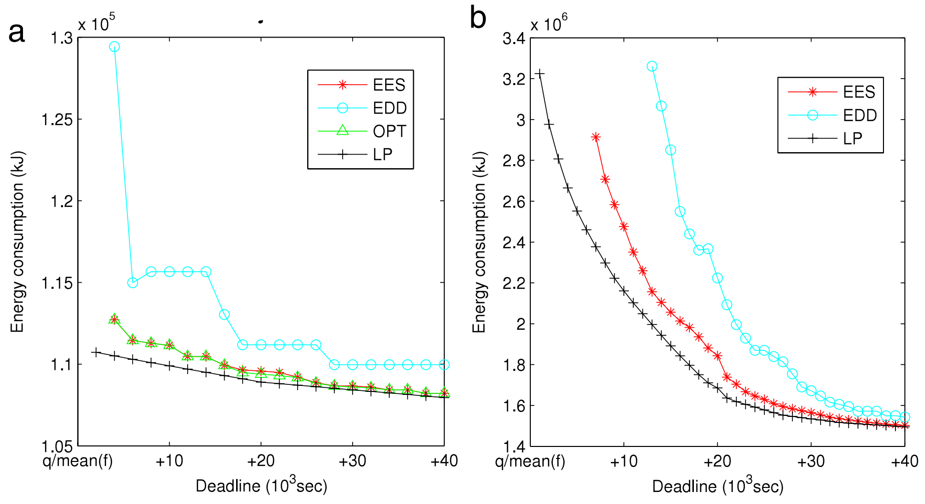

Fig. 5. The relationship between energy consumption and deadline for the same instance (a) in small-scale system and (b) in large-scale system.

also concluded to have the fluctuation in energy reduction with the decrease of the percentage, especially for the tight deadline. As shown in the table, EDD thus shows limited space for energy saving when reducing the percentage of tight deadlines, while EES exhibits the stable influence of deadline on energy consumption. For the DAS-2 workload, EES is on average 1.89% worse than OPT in energy consumption for different settings of the deadline.

We also discuss the results on deadline satisfaction for different percentages of tight deadlines. Table 6 shows the average proportion of the tasks that cannot satisfy their deadlines. EES has a small difference with OPT while performing 11.4%–96% better than EDD in deadline satisfaction.

7.3. Online EES

We extend the proposed algorithm to the online scenario in Section 6.1 and its performance also needs validation. We utilize the first week of ANL Intrepid trace (Jan 2009–Sep 2009) from PWA as our realistic workload. Since the workload log does not contain data about feedback and dependencies among tasks, we view a record as a scheduling unit. From the log, the arrival time of each task is extracted. We get the size and assigned deadline in the way described in Section 7.1. For the system, we still use the parameters in Table 1 and the system is composed of $30\mathcal{P}_1$.

For the ANL Intrepid workload, we first uniformly assign a loose deadline whose ratio is [10, 14] to the workload trace and we know that the online EES algorithm is 15.6% better than the EDF algorithm in energy reduction. When we assign a tight deadline whose deadline is [2, 6], the online algorithm is 36.7% better in energy reduction and 11.2% better in deadline satisfaction. After giving two extreme cases of deadline assignment, Fig. 6 shows the total energy consumption and deadline violation rate with the percentage of tasks who have tight deadlines. The ONEES (online EES) algorithm on average outperforms the EDF algorithm 25.72% in energy saving and 22.14% in deadline satisfaction excluding the zero values for the realistic workload.

Table 5
Average LP-normalized energy consumption.

	The percentage of tight deadlines					
	0%	20%	40%	60%	80%	100%
EDD	1.086	1.113	1.204	1.251	1.388	1.392
EES	1.014	1.032	1.059	1.096	1.107	1.127
OPT	1.009	1.021	1.039	1.064	1.081	1.100

Table 6
Average proportion of deadline violation.

	The percentage of tight deadlines					
	0%	20%	40%	60%	80%	100%
EDD (%)	1.5	12.8	19.2	36.0	61.9	93.1
EES (%)	0	0.5	3.7	15.8	32.3	82.4
OPT (%)	0	0.2	3.6	12.2	29.5	80.3

Market model. In Section 5.3, we propose a pricing scheme based on the unitcost metric, while in Section 6.3 we analyze its practical significance. Next we use experiments to validate its viability.

We first give how task deadline and task size affect unitcost using the synthetic workload. Fig. 7 shows the relationship between unitcost and task deadline and size. Unitcost will decrease with the increase of the ratio between deadline and size. When deadline is small for the fixed size, there can exist some fluctuations as shown in Fig. 7(b), but the general decreasing trend is observed. It is also observed that, when the deadline/size value is larger than 0.8, the unitcost is stable. We also explore the relationship between unitcost and the ratio of deadline/runtime using the realistic workload of ANL Intrepid trace. As shown in Fig. 8, with the increase of the ratio, the unitcost is reducing quickly. When the ratio gets beyond a certain value, its unitcost will remain constant.

As shown in Figs. 7 and 8, the monetary cost of a task may fluctuate when increasing its deadline, especially for the tight

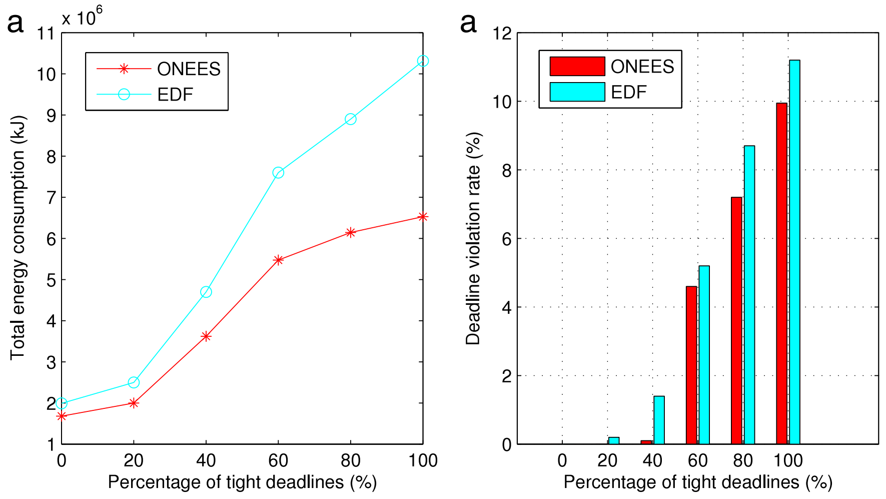

Fig. 6. The comparison on energy consumption and deadline violation rate for ONEES and EDF algorithms.

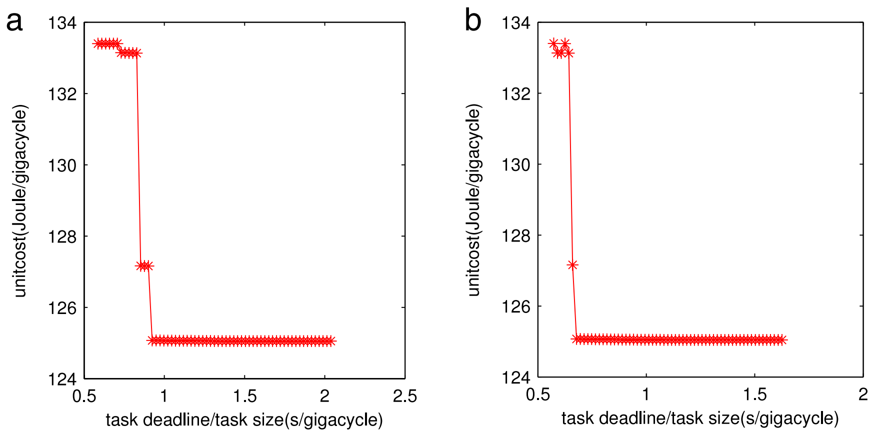

Fig. 7. The two kinds of relationship between unitcost and task deadline/task size for 50-tasks synthetic application.

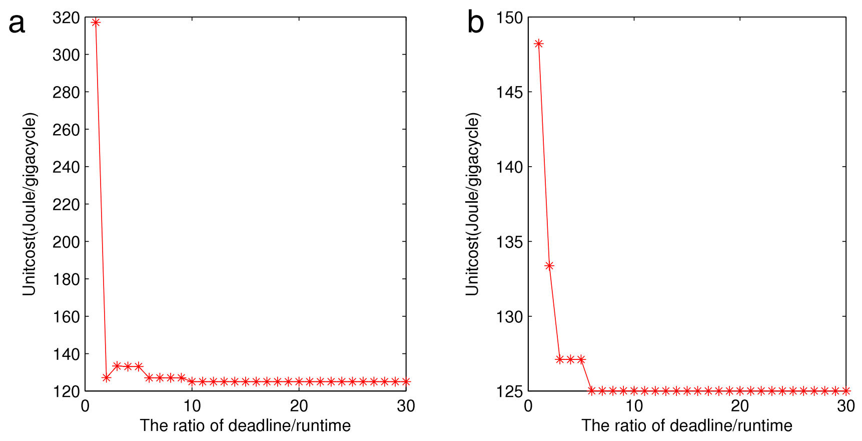

Fig. 8. The two kinds of relationship between unitcost and ratio of deadline/runtime for realistic application.

deadline. We deduce from the proposed algorithms that if the deadline of a task is tight they may have to pay more money and it is wise for tasks to use a loose deadline. We next validate the probability of the rule using the experimental samples. For each scale of workloads, we generate 500 synthetic instances and add the other 500 instances from DAS-2 workload traces. Each instance runs twice by changing the deadline of a task and the deadline is increased to 1.2, 1.4, 1.6, 1.8, and 2 times the originally tight one, respectively. We then get the sample probability of cost reduction when increasing the deadline, as shown in Table 7. If a task increases its deadline, it has on average 95% probability of reducing cost, which coincides with our expectation.

Table 7
Proportion of cost reduction with deadline increase.

	Increased ratio of the deadline				
	1.2	1.4	1.6	1.8	2
Small-scale	999/1000	1000/1000	1000/1000	1000/1000	1000/1000
Medium-scale	951/1000	953/1000	956/1000	962/1000	964/1000
Large-scale	915/1000	917/1000	917/1000	918/1000	959/1000

8. Conclusions

This paper presents both the static and online Energy-Efficient Scheduling (EES) algorithm for independent tasks with deadline constraints in heterogeneous systems. In designing the EES algorithm, we introduced unitcost metric, which describes the energy consumption of unit computation amount. We have proved NP-hardness of the problem and its subproblem, and designed approximation algorithms for each of these. Our experimental results demonstrate the static EES algorithm has almost as good energy minimization and deadline satisfaction ability as the optimal solution while the online EES algorithm has a much better performance than the EDF algorithm. Moreover, since we relate monetary cost with energy consumption through the unitcost metric and pricing scheme, both user and provider can try to control their own parameters to maximize the respective interests, which has good marketization application. As future work, we are planning to study the power-aware deadline scheduling of communication-intensive applications for heterogeneous systems.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under grant no. 61070017. The authors are grateful for the anonymous reviewers who made constructive comments.

Appendix A. Notations and analyses

For the convenience of proof description in Theorem 3, we first give the definition and analyses of several notations.

Notation 1. Before task merging, the task with the longest slack time is denoted as $T(s_{\max})$, computed as Eq. (A.1). The largest slack time is denoted as $s_{\max}$. Its size is denoted as $q(s_{\max})$:

$$T(s_{\max}) = \arg\max_{T_j \in \mathcal{T}} s_j. \tag{A.1}$$

Notation 2. Before task merging, the task with the largest computation amount is denoted as $T(q_{\max})$, computed as Eq. (A.2). The largest computation amount is denoted as $q_{\max}$. Its slack time is denoted as $s(q_{\max})$:

$$T(q_{\max}) = \arg\max_{T_j \in \mathcal{T}} q_j. \tag{A.2}$$

Notation 3. Before task merging, the task with the smallest computation amount is denoted as $T(q_{\min})$, computed as Eq. (A.3). The smallest computation amount is denoted as $q_{\min}$:

$$T(q_{\min}) = \arg\min_{T_j \in \mathcal{T}} q_j. \tag{A.3}$$

Notation 4. After task merging, the task group executed on the same resource node is called the consolidated task group, denoted as g_k $(k = 1, \dots, l)$. Their corresponding computation amounts are denoted as Q_k:

$$Q_k = \sum_{T_j \in g_k} q_j, \qquad \bigcup_k g_k = \mathcal{T}, \quad g_k \cap g_i = \emptyset, \quad \forall i, k,\ i \neq k. \tag{A.4}$$

Notation 5. For any given consolidated task group g_k, there must be a task with the longest slack time. The task is denoted as $T(s^k_{\max})$, computed as Eq. (A.5). Its slack time is denoted as $s^k_{\max}$. In the consolidated task group, there are some tasks that are put into the slack time. Let g^k_s denote these tasks, while the other tasks including $T(s^k_{\max})$ are denoted as g^k_t. They satisfy Eq. (A.6) and their sizes are denoted as $Q(g^k_s)$ and $Q(g^k_t)$:

$$T(s^k_{\max}) = \arg\max_{T_j \in g_k} s_j \tag{A.5}$$

$$g^k_s \cap g^k_t = \emptyset, \qquad g^k_s \cup g^k_t = g_k. \tag{A.6}$$

Notation 6. For any given consolidated task group g_k, there must be such a task called the dominant task. When the other tasks merge with it, they will all use its slack time. Let $T(s^k)$ denote the dominant task, and s^k is slack time. Eq. (A.7) shows their relationship. In order to describe conveniently, we use $g^k_r = g_k \setminus \{T(s^k)\}$ to denote the other tasks in g_k, and its size is denoted as $Q(g^k_r)$:

$$\sum_{T_j \in g_k, T_j \neq T(s^k)} q_j / f_0 \leq s^k. \tag{A.7}$$

Appendix B. Approximation proof

Theorem 3. $\tilde{Q} \geq Q^* - \min\{s_{\max} f_0, 2q_{\max} - q_{\min}\}$.

Proof. (i) First prove $\tilde{Q} \geq Q^* - s_{\max} f_0$.

Suppose the largest task group obtained by the OPT′ algorithm is denoted as g^*, all the other tasks that merged with dominant task $T(s^*)$ are denoted as $g^*_r = g^*/\{T(s^*)\}$. According to Eqs. (A.7) and (A.1), we get Eq. (B.1), where $Q(x)$ denotes the computation amount of task group x, s^* denotes the slack time of the dominant task, and $s_{\max}$ denotes the largest slack time in $\mathcal{T}$:

$$Q(g^*_r) \leq s^* f_0 \leq s_{\max} f_0. \tag{B.1}$$

Moreover, for task $T(s^*)$, the following is definitely right, where $q(s^*)$ is the size of the dominant task:

$$q(s^*) \leq q_{\max}. \tag{B.2}$$

According to Eqs. (B.1) and (B.2), we get

$$Q^* = Q(g^*_r) + q(s^*) \leq s_{\max} f_0 + q_{\max}. \tag{B.3}$$

For largest task group $\tilde{g}$ obtained from the LSFP algorithm, we know

$$\tilde{Q} \geq q_{\max}. \tag{B.4}$$

Otherwise, it will contradict the LSFP algorithm. If task $T(q_{\max})$ is in $\tilde{g}$, Eq. (B.4) is tenable; else it is in another smaller task group than $\tilde{g}$, and Eq. (B.4) also stands.

Therefore, according to Eqs. (B.3) and (B.4), we get

$$\tilde{Q} \geq Q^* - s_{\max} f_0. \tag{B.5}$$

(ii) Second prove $\tilde{Q} \geq Q^* - (2q_{\max} - q_{\min})$.

Suppose inequality $Q(g_r^*) - Q(g_s^1) > q_{\max}$ stands, where g_r^* denotes the remaining tasks except dominant task $T(s^*)$ in g^*, g_s^1 denotes the tasks put into the slack time of task $T(s_{\max}^1)$ in g_1 and g_1 is the first consolidated task group obtained by the LSFP algorithm, we know

$$Q(g_r^*)/f_0 - Q(g_s^1)/f_0 > q_{\max}/f_0. \tag{B.6}$$

Since non-ascending order of slack time in the LSFP algorithm guarantees $s_1 = s_{\max}$, we obtain

$$s_{\max}^1 = s_1 \geq s^*. \tag{B.7}$$

Eqs. (B.6) and (B.7) show that there must exist one task $T_k \in g_r^*$, where g_1 still has slack time to merge T_k, while LSFP makes sure that g_1 has transferred all the tasks to perform merging and there is no such task that should be consolidated but was not. Therefore, since the two cases are contradictive, we get

$$Q(g_r^*) - Q(g_s^1) \leq q_{\max}. \tag{B.8}$$

According to Eqs. (B.8) and (A.6), we get

$$Q^* - q(s^*) - Q_1 + Q(g_t^1) \leq q_{\max}. \tag{B.9}$$

Since we know $q(s^*) \leq q_{\max}$ and $Q(g_t^1) \geq q_{\min}$, Eq. (B.10) is derived:

$$Q^* - Q_1 - q_{\max} + q_{\min} \leq q_{\max}. \tag{B.10}$$

The $\tilde{g} = \arg\max_{g_k} Q_k$ of the LSFP algorithm makes sure that

$$\tilde{Q} \geq Q_1. \tag{B.11}$$

Therefore, combining Eqs. (B.10) and (B.11), we get $\tilde{Q} \geq Q^* - (2q_{\max} - q_{\min})$. □

Appendix C. Pseudocodes

The idea of the online algorithms is similar to their static cases. In this appendix we give the pseudocodes of the online algorithms, as shown in Algorithms 3 and 4.

Algorithm 3 The online EES algorithm

```
procedure ONEES($\mathcal{T}_{new}$, $\mathcal{G}$, $\mathcal{P}$, $t_{cur}$)    ▷ new arrived tasks $\mathcal{T}_{new}$, old tasks $\mathcal{G} = \{g^i\}, \forall i, P_i \in \mathcal{P}$, resource set $\mathcal{P}$, current scheduling time $t_{cur}$
    while $\mathcal{T}_{new} \neq \emptyset$ do
        $P_0 \leftarrow \arg\min_{i:P_i \in \mathcal{P}} \{p_i/f_i\}$
        if $P_0$ is off then
            LSFP($\mathcal{T}_{new}, f_0$)
        else
            ONLSFP($\mathcal{T}_{new}, g^0, f_0, t_{cur}$)
        Map $\tilde{g}$ on $P_0$
        $\mathcal{T}_{new} \leftarrow \mathcal{T}_{new} \backslash \tilde{g}$
        $\mathcal{P} \leftarrow \mathcal{P} \backslash \{P_0\}$
    Remove idle time for each task group
```

Algorithm 4 The online LSFP algorithm

```
procedure ONLSFP($\mathcal{T}_{new}, g, f, t_{cur}$)    ▷ new arrived tasks $\mathcal{T}_{new}$, old unfinished tasks $g$ on the node, node speed $f$, current scheduling time $t_{cur}$
    **update the parameters in $g$, including $t_i, s_i, S_0(d_i)$**
    for $i$ s.t. $T_i \in g$ do
        $t_i = q_i/f$
        $s_i = d_i - t_i - t_{cur}$
        $S_0(d_i) = d_i - t_{cur}$
    Sort $g$ in the non-ascending order of $s_i$, i.e., $s_i \geq s_{i'}$ for $i < i'$
    for $j$ s.t. $T_j \in g$ do    ▷ adjust $S_0(d_i)$ in $g$
        for $i$ s.t. $T_i \in g$ and $j < i$ do
            if $d_i \geq d_j$ then
                $S_0(d_i) = S_0(d_i) - t_j$
            else if $d_i > S_0(d_j) - t_j$ then
                $S_0(d_i) = S_0(d_j) - t_j$
    **update the parameters in $\mathcal{T}_{new}$, including $t_i, d_i, s_i$**
    for $i$ s.t. $T_i \in \mathcal{T}_{new}$ do
        $t_i = q_i/f$
        $d_i = d_i^{r'} + t_{cur}$
        $s_i = d_i - t_i - t_{cur}$
    **find the largest task group merged with $g$ within deadline satisfaction**
    $k \leftarrow 1$
    $\mathcal{T}'_{new} \leftarrow \mathcal{T}_{new}$
    while $\mathcal{T}'_{new} \neq \emptyset$ do
        $g_k \leftarrow g$    ▷ update the initial task group as $g$
        $Q_k \leftarrow sum(g)$
        Sort $\mathcal{T}'_{new}$ in the non-ascending order of $s_i$, i.e., $s_i \geq s_{i'}$ for $i < i'$
        $S_k(d_i) \leftarrow d_i - t_{cur}, \forall i$ s.t. $T_i \in \mathcal{T}'_{new}$
        for $j$ s.t. $T_j \in g$ do    ▷ adjust $S_k(d_i)$ in $\mathcal{T}'_{new}$ based on $g$
            for $i$ s.t. $T_i \in \mathcal{T}'_{new}$ do
                if $d_i \geq d_j$ then
                    $S_k(d_i) = S_k(d_i) - t_j$
                else if $d_i > S_0(d_j) - t_j$ then
                    $S_k(d_i) = S_0(d_j) - t_j$
        for $j$ s.t. $T_j \in \mathcal{T}'_{new}$ do
            if $S_k(d_j) \geq t_j$ then    ▷ merge $g$ with tasks in $\mathcal{T}'_{new}$
                $g_k$=$g_k \cup \{T_j\}$
                $\mathcal{T}'_{new} = \mathcal{T}'_{new} \backslash \{T_j\}$
                $Q_k = Q_k + q_j$
                for $i$ s.t.$T_i \in \mathcal{T}'_{new}$ and $j < i$ do    ▷ adjust $S_k(d_i)$ based on the new merger
                    if $d_i \geq d_j$ then
                        $S_k(d_i) = S_k(d_i) - t_j$
                    else if $d_i > S_k(d_j) - t_j$ then
                        $S_k(d_i) = S_k(d_j) - t_j$
        $k \leftarrow k + 1$
    $\tilde{g} \leftarrow \arg\max_{g_k} Q_k \backslash g$
    return $\tilde{g}$
```

References

[1] Faraz Ahmad, T.N. Vijaykumar, Joint optimization of idle and cooling power in data centers while maintaining response time, in: Proc. of 14th Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS, 2010, pp. 243–256.
[2] Cosimo Anglano, Massimo Canonico, Fault-tolerant scheduling for bag-of-tasks grid applications, in: Proc. of the 2005 European Grid Conference, EuroGrid 2005, in: Lecture Notes in Computer Science, Springer, 2005, p. 630.
[3] Hakan Aydin, Qi Yang, Energy-aware partitioning for multiprocessor real-time systems, in: Proc. of Parallel and Distributed Processing Symposium, IPDPS, 2003.
[4] Abdul Aziz, Hesham El-Rewini, Power efficient scheduling heuristics for energy conservation in computational grids, J. Supercomput. 57 (1) (2011) 65–80.

[5] Olivier Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, Loris Marchal, Yves Robert, Centralized versus distributed schedulers for multiple bag-of-task applications, IEEE Trans. Parallel Distrib. Syst. 19 (5) (2008) 698–709.

[6] Alessandro Bogliolo, Luca Benini, Ro Bogliolo, Giovanni De Micheli, A survey of design techniques for system-level dynamic power management, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 8 (3) (2000) 299–316.

[7] Paul Bratley, Michael Florian, Pierre Robillard, Scheduling with earliest start and due date constraints, Nav. Res. Logist. Q. 18 (4) (1971) 511–519.

[8] Tom Budnik, Brant Knudson, Mark Megerian, Sam Miller, Mike Mundy, Will Stockdell, Blue Gene/Q resource management architecture, 2010. http://www.green500.org/lists/2010/11/little/list.php.

[9] Jennifer Burge, Partha Ranganathan, Janet L. Wiener, Cost-aware scheduling for heterogeneous enterprise machines (cash'em), in: Proc. of 1st International Workshop on Green Computing, GreenCom, 2007.

[10] Kirk W. Cameron, Rong Ge, XiZhou Feng, High-performance, power-aware distributed computing for scientific applications, Computer 38 (11) (2005) 40–47.

[11] Ho Leung Chan, Joseph Wun Tat Chan, Tak Wah Lam, Lap Kei Lee, Kin Sum Mak, Prudence W.H. Wong, Optimizing throughput and energy in online deadline scheduling, ACM Trans. Algorithms 6 (1) (2009) 1–10.

[12] Hua Chen, Albert Mo Kim Cheng, Ying Wei Kuo, Assigning real-time tasks to heterogeneous processors by applying ant colony optimization, J. Parallel Distrib. Comput. 71 (1) (2011) 132–142.

[13] Edward T.H. Chu, Tai Yi Huang, Yu Che Tsai, An optimal solution for the heterogeneous multiprocessor single-level voltage-setup problem, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 28 (11) (2009) 1705–1718.

[14] Walfredo Cirne, Francisco Brasileiro, Jacques Sauvé, Nazareno Andrade, Daniel Paranhos, Elizeu Santos-neto, Raissa Medeiros, Federal Campina Gr., Grid computing for bag of tasks applications, in: Proc. of 3rd IFIP Conference on E-Commerce, E-Business and E-Government, 2003.

[15] Fabrício A.B. da Silva, Sílvia Carvalho, Eduardo R. Hruschka, A scheduling algorithm for running bag-of-tasks data mining applications on the grid, Lecture Notes in Comput. Sci. 3149 (1) (2004) 254–262.

[16] Fabrício A.B. da Silva, Walfredo Cirne, Francisco Vilar Brasileiro, Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids, Lecture Notes in Comput. Sci. 2790 (1) (2003) 169–180.

[17] Daniel Fabrício da Silva, Hermes Senger, Improving scalability of bag-of-tasks applications running on master–slave platforms, J. Parallel Comput. 35 (1) (2009) 57–71.

[18] Ewa Deelman, Dennis Gannon, Matthew Shields, Ian Taylor, Workflows and e-science: an overview of workflow system features and capabilities, Future Gener. Comput. Syst. 25 (5) (2009) 528–540.

[19] Michael L. Dertouzos, Control robotics: the procedural control of physical process, in: Proc. of IFIP Congress, 1974, pp. 807–813.

[20] Nikolaos D. Doulamis, Anastasios D. Doulamis, Emmanouel A. Varvarigos, Theodora A. Varvarigou, Fair scheduling algorithms in grids, IEEE Trans. Parallel Distrib. Syst. 18 (11) (2007) 1030–1048.

[21] Dror Feitelson. Parallel workloads archive, http://www.cs.huji.ac.il/labs/parallel/workload/, 2009.

[22] Bing Du, Chun Ruan, Robust performance modelling and scheduling of distributed real-time systems, J. Supercomput. 53 (1) (2010) 122–137.

[23] Electricity price,
http://money.163.com/11/0404/08/70PJ2AOE002526O3.html, 2011.

[24] Xiaobo Fan, Wolf Dietrich Weber, Luiz Andre Barroso, Power provisioning for a warehouse-sized computer, in: Proc. of 34th International Symposium on Computer Architecture, ISCA, 2007.

[25] Wuchun Feng, ChungHsing Hsu, The origin and evolution of green destiny, in: IEEE Cool Chips VII: An International Symposium on Low-Power and High-Speed Chips, 2004.

[26] Jose Jesus Fernandez, Dan Gordon, Rachel Gordon, Efficient parallel implementation of iterative reconstruction algorithms for electron tomography, J. Parallel Distrib. Comput. 68 (5) (2008) 626–640.

[27] Vincent W. Freeh, Nandini Kappiah, David K. Lowenthal, Tyler K. Bletsch, Just-in-time dynamic voltage scaling: exploiting inter-node slack to save energy in mpi programs, J. Parallel Distrib. Comput. 68 (9) (2008) 1175–1185.

[28] Michael R. Garey, David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman & Co., 1979.

[29] Saurabh Kumar Garg, Chee Shin Yeo, Arun Anandasivam, Rajkumar Buyya, Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers, J. Parallel Distrib. Comput. 71 (1) (2011) 732–749.

[30] Rong Ge, Xizhou Feng, Wuchun Feng, Kirk W. Cameron, CPU MISER: a performance-directed, run-time system for power-aware clusters, in: Proc. of International Conference on Parallel Processing, ICPP, 2007, pp. 18–25.

[31] Grid workloads archive, http://gwa.ewi.tudelft.nl/pmwiki/pmwiki.php?n=Workloads.Overview, 2007.

[32] Jianjun Han, Qinghua Li, Dynamic power-aware scheduling algorithms for real-time task sets with fault-tolerance in parallel and distributed computing environment, in: Proc. of 19th International Parallel and Distributed Processing Symposium, IPDPS, 2005, pp. 41–46.

[33] Xin Han, Tak Wah Lam, Lap Kei Lee, Isaac K.K. To, Prudence W.H. Wong, Deadline scheduling and power management for speed bounded processors, Theor. Comput. Sci. 411 (40–42) (2010) 3587–3600.

[34] Cheng He, Yixun Lin, Jinjiang Yuan, A note on the single machine scheduling to minimize the number of tardy jobs with deadlines, European J. Oper. Res. 201 (3) (2010) 966–970.

[35] Fengping Hu, Jeffrey J. Evans, Power and environment aware control of beowulf clusters, Cluster Comput. 12 (3) (2009) 299–308.

[36] IBM Blue Gene team, Overview of the IBM Blue Gene/P project, IBM J. Res. Dev. 52 (1) (2008) 199–220.

[37] Alexandru Iosup, Ozan Sonmez, Shanny Anoep, D. Epema, The performance of bags-of-tasks in large-scale distributed systems, in: Proc. of 17th International Symp on High Performance Distributed HPDC, 2008, pp. 97–108.

[38] Sandy Irani, Sandeep Shukla, Rajesh Gupta, Algorithms for power savings, ACM Trans. Algorithms 3 (4) (2007) 1–41.

[39] David E. Irwin, Laura E. Grit, Jeffrey S. Chase, Balancing risk and reward in a market-based task service, in: Proc. of 13th IEEE International Symp on High Performance Distributed Computing, HPDC, 2004.

[40] James R. Jackson, Scheduling a production line to minimize maximum tardiness, Technical Report, Management Science Research Project, Univ. of Calif., Los Angeles, 1955.

[41] Niraj K. Jha, Low power system scheduling, synthesis and displays, IEE Proceedings of Computers & Digital Techniques 152 (3) (2005) 344–352.

[42] Mohammad Kalantari, Mohammad Kazem Akbari, A parallel solution for scheduling of real time applications on grid environments, Future Gener. Comput. Syst. 25 (7) (2009) 704–716.

[43] A. Karabuto, Hdd diet: power consumption and heat dissipation, 9, 2007. http://ixbtlabs.com/articles2/storage/hddpower.html.

[44] Kyong Hoon Kim, Rajkumar Buyya, Jong Kim, Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters, in: Proc. of 7th International Cluster Computing and the Grid, CCGRID, 2007, pp. 541–548.

[45] Cynthia B. Lee, Allan E. Snavely, Precise and realistic utility functions for user-centric performance analysis of schedulers, in: Proc. of 16th International Symp on High Performance Distributed Computing, HPDC, 2007, pp. 107–116.

[46] Young Choon Lee, Albert Y. Zomaya, Practical scheduling of bag-of-tasks applications on grids with dynamic resilience, IEEE Trans. Comput. 56 (6) (2007) 815–825.

[47] Young Choon Lee, Albert Y. Zomaya, Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling, in: Proc. of 9th IEEE/ACM International Cluster Computing and the Grid, CCGRID, 2009, pp. 92–99.

[48] Charles Lefurgy, Xiaorui Wang, Malcolm Ware, Server-level power control, in: Proc. of International Conference on Autonomic Computing, ICAC, 2007, pp. 11–15.

[49] Keqin Li, Performance analysis of power-aware task scheduling algorithms on multiprocessor computers with dynamic voltage and speed, IEEE Trans. Parallel Distrib. Syst. 19 (11) (2008) 1484–1497.

[50] Chunlin Li, Layuan Li, Joint optimisation of application QoS and energy conservation in grid environment, Internat. J. Systems Sci. 41 (9) (2010) 1027–1041.

[51] Minming Li, Becky J. Liu, Frances F. Yao, Min-energy voltage allocation for tree-structured tasks, J. Comb. Optim. 11 (3) (2006) 305–319.

[52] Minming Li, Frances F. Yao, An efficient algorithm for computing optimal discrete voltage schedules, SIAM J. Comput. 35 (3) (2005) 658–671.

[53] Ming Hong Lin, Adam Wierman, Lachlan L.H. Andrew, Eno Thereska, Dynamic right-sizing for power-proportional data centers, in: Proc. of 30th IEEE International Conference on Computer Communications, IEEE INFOCOM, 2011.

[54] Cong Liu, Sanjeev Baskiyar, A general distributed scalable grid scheduler for independent tasks, J. Parallel Distrib. Comput. 69 (3) (2009) 307–314.

[55] Yan Ma, Bin Gong, Lida Zou, Marginal pricing based scheduling strategy of scientific workflow using cost-gradient metric, in: Proc. of 8th International Conference on Grid and Cooperative Computing, GCC, 2009, pp. 136–143.

[56] Yan Ma, Bin Gong, Lida Zou, Energy-optimization scheduling of task dependent graph on DVS-enabled cluster system, in: Proc. of ChinaGrid Annual Conference, 2010, pp. 183–190.

[57] David Meisner, Brian T. Gold, Thomas F. Wenisch, PowerNap: eliminating server idle power, in: Proc. of 14th Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS, 2009, pp. 205–216.

[58] Tridib Mukherjee, Ayan Banerjee, Georgios Varsamopoulos, Sandeep K.S. Gupta, Sanjay Rungta, Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers, Comput. Netw. 53 (17) (2009) 2888–2904.

[59] Marco A.S. Netto, Rajkumar Buyya, Coordinated rescheduling of bag-of-tasks for executions on multiple resource providers, Technical Report, 2010.

[60] Xiaojun Ruan, Xiao Qin, Ziliang Zong, Kiranmai Bellam, Mais Nijim, An energy-efficient scheduling algorithm using dynamic voltage scaling for parallel applications on clusters, in: Proc. of 16th International Conference on Computer Communications and Networks, ICCCN, 2007, pp. 735–740.

[61] John A. Stankovic, Marco Spuri, Marco Di Natale, Giorgio Buttazzo, Implications of classical scheduling results for real-time systems, Computer 28 (6) (1995) 16–25.

[62] J.R. Stiles, T.M. Bartol, E.E. Salpeter, M.M. Salpeter, Monte Carlo simulation of neuromuscular transmitter release using MCell, a general simulator of cellular physiological processes, Comput. Neurosci. (1998) 279–284.

[63] Riky Subrata, Albert Y. Zomaya, Bjorn Landfeldt, Cooperative power-aware scheduling in grid computing environments, J. Parallel Distrib. Comput. 70 (2) (2010) 84–91.
[64] Top 500, http://www.top500.org/lists/2010/11/press-release, 2010.
[65] Li Ya Tseng, Yeh Hao Chin, Shu Ching Wang, A minimized makespan scheduler with multiple factors for grid computing systems, Expert Syst. Appl. 36 (8) (2009) 11118–11130.
[66] Mark Weiser, Brent Welch, Alan Demers, Scott Shenker, Scheduling for reduced CPU energy, in: Proc. of 1st USENIX Symp. Operating Systems Design and Implementation, OSDI, 1994, pp. 13–23.
[67] Chu Liang Weng, Xin Da Lu, Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid, Future Gener. Comput. Syst. 21 (1) (2005) 271–280.
[68] Adianto Wibisono, Zhiming Zhao, Adam Belloum, Marian Bubak, A framework for interactive parameter sweep applications, in: Proc. of 8th Symp Cluster Computing and the Grid, CCGRID, 2008, pp. 481–490.
[69] Guowei Wu, Zichuan Xu, Temperature-aware task scheduling algorithm for soft real-time multi-core systems, J. Syst. Softw. 83 (12) (2010) 2579–2590.
[70] Frances Yao, Alan Demers, Scott Shenker, A scheduling model for reduced CPU energy, in: Proc. of 36th IEEE Symp on Foundations of Computer Science, FOCS, 1995, pp. 374–382.
[71] Ziliang Zong, Energy-efficient resource management for high-performance computing platforms, Ph.D. Thesis, Department of Computer Science and Software Engineering, Auburn University, 2008.

Yan Ma is a Ph.D. candidate in the School of Computer Science and Technology from Shandong University and also a joint Ph.D. student in the Department of Computer Science and Engineering, University of California, San Diego. Her research interests are in the areas of high performance computing and power-aware computing.

Bin Gong is a professor in the School of Computer Science and Technology, Dean of the Computing Center at Shandong University, and the Vice Dean in the High-Performance Computing Center of Shandong Province. He has completed and participated in National 863 Research Programs, National Natural Science Foundation Programs. His research interests include grid computing, high performance computing, cluster computing, scheduling, and power-aware computing.

Ryo Sugihara is a postdoctoral scholar at the Department of Computer Science and Engineering of the University of California, San Diego. He received his Ph.D. (2009) in computer science from UCSD and M.Eng. (1999) and B.Eng. (1997) in mechano-informatics from the University of Tokyo, Japan. Previously he has worked as a researcher at IBM Research—Tokyo. His research interests include scheduling, sensor networks and mobile ad hoc networks.

Rajesh Gupta is a professor and holder of the QUALCOMM endowed chair in Embedded Microsystems in the Department of Computer Science and Engineering at UC San Diego, California. He leads the Microelectronic Embedded Systems Lab and is head of the Embedded Systems Group at UCSD. Rajesh did his undergraduate education at IIT-Kanpur and his graduate education at UC Berkeley and Stanford. He currently serves as an advisor to Tallwood Venture Capital, RealIntent, Calypto and Packet Digital Corporation. His current research is energy efficient and mobile computing.