Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

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

# Adaptive energy-efficient scheduling for real-time tasks on DVS-enabled heterogeneous clusters

Xiaomin Zhu [a,*], Chuan He [a], Kenli Li [b], Xiao Qin [c]

[a] Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha 410073, PR China
[b] School of Computer and Communication, Hunan University, Changsha 410082, PR China
[c] Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849-5347, USA

ARTICLE INFO

*Article history:*
Received 24 February 2011
Received in revised form
27 June 2011
Accepted 9 March 2012
Available online 20 March 2012

*Keywords:*
Cluster
Real-time
Scheduling
Energy-efficient
Adaptivity
Dynamic voltage scaling

ABSTRACT

Developing energy-efficient clusters not only can reduce power electricity cost but also can improve system reliability. Existing scheduling strategies developed for energy-efficient clusters conserve energy at the cost of performance. The performance problem becomes especially apparent when cluster computing systems are heavily loaded. To address this issue, we propose in this paper a novel scheduling strategy – adaptive energy-efficient scheduling or AEES – for aperiodic and independent real-time tasks on heterogeneous clusters with dynamic voltage scaling. The AEES scheme aims to adaptively adjust voltages according to the workload conditions of a cluster, thereby making the best trade-offs between energy conservation and schedulability. When the cluster is heavily loaded, AEES considers voltage levels of both new tasks and running tasks to meet tasks' deadlines. Under light load, AEES aggressively reduces the voltage levels to conserve energy while maintaining higher guarantee ratios. We conducted extensive experiments to compare AEES with an existing algorithm – MEG, as well as two baseline algorithms – MELV, MEHV. Experimental results show that AEES significantly improves the scheduling quality of MELV, MEHV and MEG.

## 1. Introduction

In the past decade, computer cluster computing systems have increasingly become a popular cost-effective platform to run computing-intensive and data-intensive real-time applications like real-time weather forecast, image processing, signal processing, and stock trading [4,15,38,6]. Much of this trend can be attributed to the rapid enhancements in processing power of commodity-off-the-shelf hardware components and their low cost. The TOP500 supercomputer list made on June 2010 showed that clusters accounted for 84.8% within the fastest 500 computers in the world [26], indicating that a majority of high-performance supercomputers are built using inexpensive personal computers connected by high-speed networks. Especially, heterogeneous clusters have received a good deal of attention, because a homogeneous cluster becomes a heterogeneous one after newly purchased hardware components are incorporated into the cluster with old components. With the advance of cloud computing, heterogeneous clusters are widely applied in modern data and supercomputing centers. In this study we focus on heterogeneous clusters, which significantly affect the global scientific and business communities [18].

Although clusters are able to provide high-performance computing solutions, large-scale clusters consume tremendous amount of energy. For example, the total power of a 360-Tflops high-performance cluster (e.g. IBM Blue Gene/L) with traditional processors could exceed 10 megawatts, possibly approaching 20 megawatts. The total power of a large-scale cluster is approximately equal to the amount of power used in 22,000 US households [8]. In addition, high-temperature heat dissipation caused by large-scale clusters requires cooling equipments (e.g., air conditioners) to control temperature in supercomputers and data centers. The primary design goal of Google's data centers is not high performance but high energy efficiency, because data centers can consume as much electricity as a city [25].

High energy consumption of clusters has three disadvantages. First, the energy cost of running clusters is very high. For example, the total energy cost of a single 200 W server (e.g., IBM 1U*300) is 180 $/year [2]. A large cluster with hundreds or thousands of computing nodes can lead to huge cost of electricity. Second, the high energy consumption has negative impacts on environment. It is evaluated that producing 1 kW electricity power needs to consume 0.4 kg coal and 4L water. Meanwhile, 0.272 kg solid powder, 0.997 kg $CO_2$, and 0.03 kg $SO_2$ are produced [21]. High

* Corresponding author.
*E-mail addresses:* xmzhu@nudt.edu.cn (X. Zhu), chuanhe@nudt.edu.cn (C. He), lkl510@263.net (K. Li), xqin@auburn.edu (X. Qin).

0743-7315/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.03.005

energy consumption leads to huge carbon dioxide emissions. If electricity used by clusters is generated by thermal power stations, the negative impacts become more pronounced. Last, high energy consumption caused by a large number of computing nodes in a cluster results in high temperature that greatly affects the system reliability. Evidence shows (the Arrhenius equation) that computing in high temperature is more error-prone than in a normal environment, because the expected failure rate of an electronic component doubles for every 10 °C increased [7]. Therefore, reducing energy consumption of computing platforms is highly indispensable; we must make future clusters energy efficient.

The dynamic voltage scaling technique or DVS is an efficient approach to reducing energy consumption [22]. DVS exploits the relationship between CPU supply voltages of devices and power usages (e.g., Crusoe [1] and ARM7D [5]). DVS has been implemented in real-world processors like Intel SpeedStep and AMD PowerNow!. Most modern processors supporting DVS use discrete voltage levels. Thus, one can save a processor's energy by degrading CPU voltage while keeping the processor to operate at a slow speed.

In addition to the DVS technique, a second approach to conserving energy is to turn off idle computing nodes. This approach is efficient if most idle periods are very large. In real-time applications where idle periods are small, turning off computing nodes not only has a high overhead, but also introduces a high latency. A similar justification can be found in a previous study [30]. In our study, we focus on dynamic scheduling of real-time tasks on heterogeneous clusters with medium and high loads. Due to high overhead, frequently turning on and off computing nodes is not an ideal solution to conserve energy for clusters supporting real-time applications. Therefore, we decide to apply the DVS technology to adjust voltages of running nodes as an energy-saving energy approach.

The correctness of real-time tasks depends not only on the logic results of computation, but also on the time instants at which these results are produced [34]. Since the DVS technique conserves a CPU's energy consumption through CPU voltage (clock frequency) reduction at the expense of increasing the execution time of real-time tasks, applying DVS to save energy for real-time tasks must carefully adjust the execution times of the real-time tasks within their timing constraints.

*Motivation*. In addition to conserving energy consumption caused by real-time tasks, improving performance (e.g., guarantee ratio) is desirable and, in many cases, mandatory for real-time applications. For example, a real-time military surveillance application requires real-time response besides significant energy conservation, because missing deadlines makes the application worthless [36]. Task scheduling strategies that only focus on reducing energy consumption are inadequate for many real-time applications. To address this problem, we attempt to incorporate adaptivity into energy-efficient scheduling schemes for real-time applications. Specifically, our approach first gives high priority to deal with schedulability when a real-time system is heavily loaded even though much energy consumption may be produced. When the system is under light load, our approach strives to reduce energy consumption while achieving high schedulability for real-time tasks.

Most existing energy-efficient scheduling algorithms do not address the adaptivity issue in the context of real-time heterogeneous clusters. This problem motivates us to design and implement a novel adaptive energy-efficient scheduling strategy for real-time tasks on DVS-enabled heterogeneous clusters.

*Contributions*. Our main contributions are summarized as follows:

(1) We constructed an energy consumption model that adequately considers the adaptivity.
(2) We developed an *a*daptive *e*nergy-*e*fficient *s*cheduling strategy or AEES for real-time tasks running on heterogeneous clusters.
(3) We conducted experimental evaluation of AEES using a simulated heterogeneous cluster.

The rest of this paper is organized as follows. The related work is summarized in Section 2. Section 3 presents the adaptive power-aware scheduling model. Section 4 describes the AEES scheduling strategy that contains two algorithms—EEGS and LVA. The main principles of EEGS and LVA are also discussed in Section 4. Simulation experiments and performance analysis are presented in Section 5. Section 6 concludes the paper with a few future directions.

## 2. Related work

Over the past decade, increasing attention has been directed toward energy conservation for high-performance clusters [18,28,37,20,13,44]. Among many energy-saving techniques, scheduling is an efficient approach to reducing energy consumption on clusters. A scheduling mechanism makes an effort to allocate tasks to computing nodes in a cluster where the nodes can be operated at the possibly lowest voltage level. Many practical scheduling algorithms developed for multiprocessors are NP-complete [32], motivating researchers to design heuristic algorithms to solve the scheduling problems.

In a broad sense, the scheduling algorithms exist in two forms: static and dynamic [17]. Static scheduling algorithms make scheduling decisions before tasks are submitted to a cluster, and are often applied to schedule periodic tasks [29,10]. However, aperiodic tasks whose arrival times are not known a priori must be handled by dynamic scheduling algorithms (see, for example, [9,27]). In this study, we focus on scheduling aperiodic and independent real-time tasks. Our algorithm can be extended to deal with tasks with precedence constraints, because dependent tasks are equivalent to independent tasks by modifying ready times and deadlines of the dependent tasks [23]. Moreover, some scheduling algorithms of real-time tasks are preemptive, whereas our scheduling algorithm is non-preemptive, i.e., an executing task cannot be preempted by another task, which is more efficient, particularly fit for soft real-time applications than the preemptive approaches because of reducing the overheads required for switching among tasks [19]. Consequently, the non-preemptive scheduling scheme is employed in this paper.

Laszewski et al. focused on reducing energy consumption of virtual machines on a homogeneous cluster using DVS-based scheduling. Zong et al. investigated a two-phase energy-efficient scheduling strategy called EETDS for dependent tasks with intensive communications. EETDS applies a duplication-based method to shrink communication energy cost when allocating parallel tasks to a heterogeneous cluster [44]. Kotla et al. employed the execution characteristics of tasks running on clusters to predict their performance at available frequency settings and to schedule these tasks with the lowest frequency [16]. Xie et al. proposed a scheduling scheme – BEATA – that considers both energy consumption and schedule length to solve the energy-latency dilemma for tasks of parallel applications in heterogeneous embedded systems [35]. Hu et al. described an approach exploiting live migration of virtual machines to transfer load among computing nodes to reduce energy consumption on homogeneous and heterogeneous clusters [11]. Although these scheduling strategies are able to achieve high performance for non-real-time applications, they are inadequate for real-time applications due to the lack of guarantee to finish real-time tasks within their deadlines.

Previous studies have incorporated energy conservation into real-time scheduling. Nélis et al. considered the problem of minimizing energy consumption caused by a set of sporadic constrained-deadline real-time tasks scheduled on a fixed number of processors. Their scheduling algorithm is preemptive; each process can start its execution on any processor and may migrate at run-time from one processor to another if it gets preempted by earlier-deadline processes [28]. Kim et al. presented two power-aware scheduling algorithms (space-shared and time-shared) for bag-of-tasks real-time applications on DVS-enabled clusters to minimize energy dissipation while meeting applications' deadlines [13]. Unfortunately, the above methods were designed for homogeneous computing environments, having no inherent capability of supporting heterogeneous systems. Energy saving on heterogeneous clusters is complicated, because a high-voltage node may yield lower energy consumption due to its higher processing speed.

Yu et al. studied the allocation problem of a set of independent tasks in a real-time system consisting of heterogeneous DVS-enabled processing elements [37]. Yu's scheduling scheme, which is static in nature, can efficiently schedule periodic real-time tasks. Wilkins et al. investigated the energy-aware task allocation problem for assigning a set of real-time tasks onto heterogeneous machines of a computational grid each equipped with DVS feature using non-cooperative game theory [33]. Liu et al. developed an algorithm or PASS for real-time tasks with different priorities and deadlines on DVS-enabled heterogeneous clusters [20]. PASS schedules both hard and soft real-time tasks. First, PASS schedules hard real-time tasks in order to meet their deadlines. Next, slack times are utilized to execute soft real-time tasks within timing constraints. PASS is a batch-style scheduling algorithm, in which submitted tasks are collected and scheduled as a meta-task when a scheduling event is triggered. Regarding the batch-style scheduling, although more tasks information can be used to make the scheduling more efficient, in real-time systems, it may result in some tasks arriving earlier miss their deadlines due to waiting delay [39]. Therefore, in our study, we employ the immediate-style scheduling, i.e., scheduling a task once it arrives.

In our previous studies [40,39,43,42], we investigated a set of scheduling strategies for real-time tasks on heterogeneous clusters. However, we did not address the energy saving issues on heterogeneous clusters. In this paper, we pay attention to adaptive energy-efficient scheduling for non-preemptive, real-time, and aperiodic tasks on DVS-enabled heterogeneous clusters. Aiming at improving energy efficiency and schedulability, our new dynamic strategy can adaptively adjust supply voltage levels according to workload conditions.

## 3. System model

This section introduces the models, notion, and terminology used throughput this paper. For future reference, we summarize notation used in this study in Table 1.

### 3.1. Energy-efficient real-time scheduling architecture

Let us concentrate on a $m$-node cluster, in which $m$ heterogeneous nodes (i.e., with different processing powers) are connected via a high-speed interconnection network (e.g., Myrinet and InfiniBand) to execute real-time tasks submitted by $n$ users. Let a heterogeneous cluster be composed of a set $N = \{n_1, n_2, \dots, n_m\}$ of heterogeneous nodes. Fig. 1 shows the architecture of an energy-efficient real-time scheduling mechanism.

The scheduler in Fig. 1 encompasses a real-time controller and an adaptive voltage controller. The real-time controller and adaptive voltage controller work together and determine if an arriving task in the global queue can be admitted or not. Once the task is accepted, a voltage level will be assigned by the scheduler.

**Table 1**
Definitions of notation.

| Notation | Definition |
|---|---|
| $n_j$ | The $j$th computing node in the node set $N = \{n_1, n_2, \dots, n_m\}$ of a cluster |
| $t_i$ | The $i$th task in the task set $T = \{t_1, t_2, \dots, t_n\}$ |
| $a_i$ | The arrival time of $t_i$ |
| $d_i$ | The deadline of $t_i$ |
| $r_j$ | The remaining execution time of the running task on $n_j$ |
| $est_{ij}$ | The earliest start time of $t_i$ on $n_j$ |
| $x_{ij}$ | $x_{ij}$ is "1" if $t_i$ is assigned to $n_j$; otherwise, $x_{ij}$ is "0" |
| $o_{ij}$ | The execution order of $t_i$ on $n_j$ |
| $w_{ij}$ | $w_{ij}$ is "1" if $t_i$ is waiting in the local queue of $n_j$, and is "0" else |
| $V_{jk}$ | The $k$th voltage level of $n_j$ |
| $v_{ij}$ | The selected voltage of $t_i$ on $n_j$, $v_{ij} \in V_j$ |
| $e_{ij}(v_{ij})$ | The execution time of $t_i$ on $n_j$ using the voltage level $v_{ij}$ |
| $e_i^{\max}$ | The maximal execution time of $t_i$ |
| $s_i$ | The task size of $t_i$ |
| $m_{jk}$ | The processing speed of $n_j$ with voltage level $v_k$ |
| $f_j(v_{ij})$ | The frequency of $n_j$ with voltage level $v_{ij}$ |
| $ec_{ij}$ | The energy consumption of $t_i$ on $n_j$ |
| $p_j^{idle}$ | The power of $n_j$ when it is idle |
| $E_{dynamic}$ | The dynamic energy consumption |
| $E_{static}$ | The static energy consumption |
| $E$ | The sum of dynamic energy consumption and static energy consumption |
| $ec^{active}(T, N, X, V)$ | The total energy consumption of nodes when they are active |
| $ec^{idle}(T, N, X, V)$ | The total energy consumption of nodes when they are idle |
| $ec(T, N, X, V)$ | The total energy consumption of nodes of a cluster |
| $EV(T, N, X, V)$ | The minimal energy consumption of a cluster |
| $AN(X)$ | The maximal number of accepted tasks |

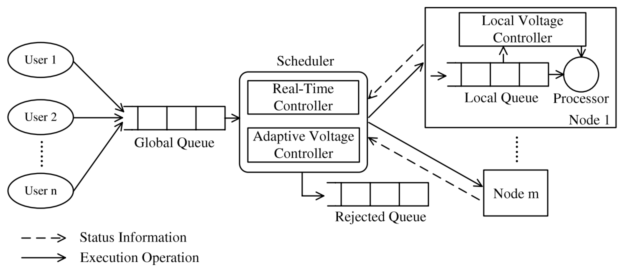

**Fig. 1.** Energy-efficient real-time scheduling architecture.

Each node in the cluster maintains a local queue in which admitted tasks are queuing up for execution on the node. The local voltage controller in each node aims at minimizing the voltage levels for admitted tasks to reduce energy consumption.

When a new task arrives, the scheduler follows three steps below to allocate and schedule the task.

*Step* 1: The scheduler checks system status information (e.g., voltage levels of the nodes, tasks running on the nodes, tasks waiting in the local queues, and actual execution times of finished tasks). It is critical to collect and manage the system status information, because schedulers largely depend on such information to choose the most energy-efficient computing nodes to run new tasks. All the information can be stored and maintained by the scheduler. The system information can be gathered in two approaches.

- First, the status information can be periodically collected. A short period (e.g., a few seconds) increases accuracy at the cost of high overhead, whereas a long period (e.g., a few minutes) reduces both monitoring overhead and accuracy. The periods, regardless of their lengths, should be adjusted by a system administrator based on application requirements.
- Second, the status information can be gathered when a new task arrives. Compared with the first approach, the second scheme

incurs lower overhead under low workload conditions. When the load becomes very high (e.g., many tasks arrive in a very short time period), the first approach is better than the second one.

It is worth noting that run-time execution conditions may be different with stored information, because execution times are estimated and computing nodes may fail due to various reasons. Fortunately, this problem is addressed by periodically collecting system status information. After the system information is gathered in a real-time manner, the system status data are immediately updated.

*Step* 2: The scheduler decides whether or not the new task can be allocated to a node and completed within its deadline by the *e*nergy-*e*fficient *g*lobal *s*cheduling algorithm or EEGS. In the process of scheduling, EEGS makes the best effort to schedule a new task to a node with the possibly lowest voltages. If the new task's deadline is not guaranteed, it will be dropped to the rejected queue. Otherwise the task will be transferred to a destination node.

*Step* 3: The status information including node voltage, sequence of the new task, execution time of tasks waiting in this node are passed on to this destination node.

After a task in one node is executed, the local voltage adjuster relies on the *l*ocal *v*oltage *a*djusting algorithm or LVA to dynamically reduce the node voltage subject to the timing constraints of tasks waiting in the local queue. The dynamic voltage scaling approach can achieve high energy efficiency of the heterogeneous clusters.

## 3.2. Task model

We consider a set $T = \{t_1, t_2, \dots, t_n\}$ of soft real-time tasks that are independent, non-preemptive, and aperiodic. There is no communication among tasks. Each task can be executed on only one node; a task cannot be partitioned and distributed across multiple nodes.

Let $a_i$ and $d_i$ be the arrival time and deadline of task $t_i$, respectively. Let $EST = (est_{ij})_{n\times m}$ be an earliest start time matrix of tasks, where element $est_{ij}$ denotes the earliest start time of task $t_i$ on node $n_j$. To model the heterogeneity of a cluster, we denote $E = (e_{ij})_{n\times m}$ as an execution time matrix of tasks, where element $e_{ij}$ denotes the execution time of task $t_i$ on node $n_j$.

The aforementioned task model largely relies on a way of estimating execution times of tasks assigned to given computing nodes. The task execution times must be approximately estimated and provided prior to scheduling tasks on clusters. A few execution-time estimation techniques are commonly used in the scheduling research community [3]. Such an execution-time estimation process is reasonable and practical, because execution times can be estimated by the code profiling and statistical prediction techniques (see, for example, [31,12]).

An allocation matrix $X = (x_{ij})_{n\times m}$ is used to reflect a mapping of $n$ tasks to $m$ nodes. Element $x_{ij}$ in $X$ is "1" if task $t_i$ is assigned to node $n_j$; otherwise, $x_{ij}$ is "0". Let $O = (o_{ij})_{n\times m}$ be an execution order matrix, where element $o_{ij}$ represents the execution order of task $t_i$ on node $n_j$. $w_{ij} = 1$ if task $t_i$ is waiting in the local queue of node $n_j$, else $w_{ij} = 0$. $f_{ij}$ is the finish time of task $t_i$ on node $n_j$.

## 3.3. Energy consumption model

It is assumed that the processor of each computing node in a cluster is DVS enabled in the sense that the processor can be operated in multiple voltages with different frequencies by CMOS circuits. The main energy consumption $E$ caused by a cluster includes two parts—dynamic energy consumption $E_{dynamic}$ due to circuit switching activities in systems and static energy consumption $E_{static}$ for the leakage currents in the circuits [13]. Thus, we have

$$E = E_{dynamic} + E_{static}. \tag{1}$$

Generally speaking, energy consumption is dominated by the dynamic energy consumption [41,24]. Although the leakage energy consumption cannot be ignored when the deep-micro technology is applied, the contribution of the leakage energy to the overall energy consumption follows a similar trend to the dynamic energy consumption. To simplify our energy consumption model, we focus on the dynamic energy consumption that can be approximated as:

$$E = \alpha f V_{dd}^2 \triangle t, \tag{2}$$

where $V_{dd}$ is supply voltage, $f$ is the number of clock cycles per second, i.e., processor clock frequency, $\triangle t$ is the execution period, and $\alpha$ is a constant. Clock frequencies are positive correlated with supply voltages. Reducing voltages can result in low clock frequencies.

Eq. (2) suggests that the energy consumption can be reduced by lowering the supply voltage at the cost of speed (i.e., slowdown of the execution time). One can save energy by degrading voltage levels and delaying the execution time of real-time tasks as long as timing constraints are met (see, for example, [13]).

We denote $V = \{V_1, V_2, \dots, V_j\}$ as a set of voltages of all nodes in a cluster. Let $V_j = \{V_{j1}, V_{j2}, \dots, V_{jk}\}$ be a set of supplied voltages for node $n_j$. For simplicity, we assume that $V_{j1} < V_{j2} < \cdots < V_{jk}$. $v_{ij} \in V_j$ is a selected voltage of task $t_i$ on node $n_j$. The execution time of task $t_i$ on node $n_j$ using voltage level $v_{ij}$ is represented as $e_{ij}(v_{ij})$. $v_{ij}$ is a feasible voltage level, which should satisfy the following two inequalities, (1) $f_{ij} \le d_i$, and (2) $V_{j1} \le v_{ij} \le V_{jk}$. We have $e_{ij}(v_{ij}) = s_i/m_{jk}$, where $s_i$ is the size of task $t_i$, and $m_{jk}$ is the processing speed of node $n_j$ with supply voltage level $v_k$. In this equation, $v_k$ equals to $v_{ij}$. Thus, the energy consumption $ec_{ij}$ of task $t_i$ on node $n_j$ can be written as follows:

$$ec_{ij} = \alpha f_j(v_{ij}) v_{ij}^2 e_{ij}(v_{ij}), \tag{3}$$

where $f_j(v_{ij})$ denotes the frequency of node $n_j$ with voltage $v_{ij}$.

Given a real-time application with task set $T$, a node set $N$, an allocation matrix $X$, and voltage set $V$, the total energy consumption of all tasks in this application is:

$$\begin{aligned} ec^{active}(T, N, X, V) &= \sum_{j=1}^{m}\sum_{i=1}^{n} x_{ij} ec_{ij} \\ &= \sum_{j=1}^{m}\sum_{i=1}^{n} x_{ij}\alpha f_j(v_{ij}) v_{ij}^2 e_{ij}(v_{ij}). \end{aligned} \tag{4}$$

Note that Eq. (4) does not incorporate energy consumption caused by idle nodes. In this study, we set the voltage to the lowest level when a node is idle, and the power of node $n_j$ is denoted as $p_j^{idle}$ when it is idle. Consequently, the total energy consumption of idle nodes is expressed as:

$$\begin{aligned} & ec^{idle}(T, N, X, V) \\ &= \sum_{j=1}^{m} p_j^{idle}\left(\max_{i=1}^{n}\{f_{ij}\} - \sum_{i=1}^{n} x_{ij} e_{ij}(v_{ij})\right) \\ &= \sum_{j=1}^{m} \alpha f_j(V_{j1}) V_{j1}^2 \left(\max_{i=1}^{n}\{f_{ij}\} - \sum_{i=1}^{n} x_{ij} e_{ij}(v_{ij})\right), \end{aligned} \tag{5}$$

where $\max_{i=1}^{n}\{f_{ij}\}$ is the finish time of task that is the latest executed on node $n_j$, and $\max_{i=1}^{n}\{f_{ij}\} - \sum_{i=1}^{n} x_{ij} e_{ij}(v_{ij})$ is the total idle time on node $n_j$.

Therefore, the total energy consumption of the nodes in a cluster is derived form Eqs. (4) and (5) as:

$$\begin{aligned} & ec(T, N, X, V) \\ &= ec^{active}(T, N, X, V) + ec^{idle}(T, N, X, V) \end{aligned}$$

$$= \sum_{j=1}^{m} \sum_{i=1}^{n} x_{ij} \alpha f_j(v_{ij}) v_{ij}^2 e_{ij}(v_{ij})$$
$$+ \sum_{j=1}^{m} \alpha f_j(V_{j1}) V_{j1}^2 \left( \max_{i=1}^{n}\{f_{ij}\} - \sum_{i=1}^{n} x_{ij} e_{ij}(v_{ij}) \right). \quad (6)$$

One of our scheduling objectives is to minimize the energy consumption of nodes in a cluster. Thus, the following energy value function needs to be minimized, subject to certain timing constraints:

$$EV(T, N, X, V) = \min \{ec(T, N, X, V)\}. \quad (7)$$

Importantly, a node running a task with the least energy consumption could result in a late finish time for the task, which in turn may affect the admissions of subsequently arriving tasks. Energy savings gained by low voltage levels can negatively affect the schedulability of clusters. In our design, we aim to maximize the number of admitted real-time tasks within a range of voltage levels of nodes. Hence, the function representing the number of accepted tasks must be maximized under timing constraints:

$$AN(X) = \max \left\{ \sum_{j=1}^{m} \sum_{i=1}^{n} x_{ij} \right\}. \quad (8)$$

Energy conservation and guarantee ratio are two conflicting objectives in the context of real-time task scheduling on a cluster. Our energy-efficient scheduling strategy AEES (see next section) makes the best trade-offs between energy saving (see Eq. (7)) and guarantee ratio (see Eq. (8)) according to the workload of a cluster. If the cluster is heavily loaded, AEES favors schedulability over energy efficiency. In contrast, when the cluster is under light load, AEES degrades the voltage levels for admitted tasks to reduce energy consumption.

## 4. The scheduling AEES strategy

We are now in a position to present in this section an adaptive energy-efficient scheduling strategy or AEES for independent aperiodic soft real-time tasks on DVS-enabled heterogeneous clusters. AEES takes the issues of schedulability, energy conservation, and system workload into account. The AEES strategy is composed of two algorithms—the EEGS algorithm and the LVA algorithm (see the two subsections below). EEGS and LVA are seamlessly integrated to adaptively adjust voltage levels of computing nodes in a cluster in accordance to the cluster's workload.

### 4.1. The EEGS algorithm

The EEGS algorithm, used to accommodate new tasks, executes the earliest deadline first (EDF) policy to improve the schedulability of real-time clusters. To facilitate the presentation of EEGS, we introduce the following two properties.

**Property 1.** *If a real-time task $t_i$ is scheduled on node $n_j$, then the following condition must be met.*

$$st_{ij} + e_{ij}(v_{ij}) \le d_i, \quad (9)$$
$$\forall t_k,\ o_{kj} > o_{ij}, \quad w_{kj} = 1 : st_{kj} + e_{kj}(v_{kj}) \le d_k, \quad (10)$$

where $st_{ij}$ and $st_{kj}$ are the start time of task $t_i$ on node $n_j$, and the start time of task $t_k$ on node $n_j$, respectively.

Property 1 states that if a task can be allocated to a node, the task must be finished before its deadline. Furthermore, waiting tasks whose execution orders are later than that of the new task in the node's local queue must be completed before their deadlines. Because the execution time of a real-time task will be increased

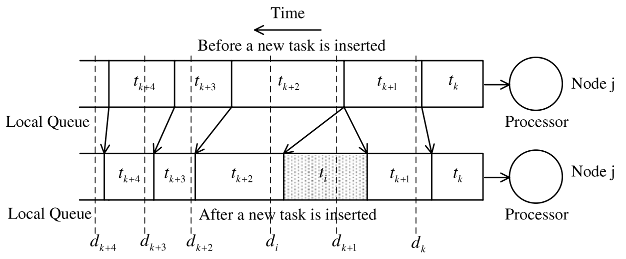

**Fig. 2.** A case of Property 2: Since task $t_i$ is admitted, the earliest deadlines for tasks $t_{k+2}$, $t_{k+3}$, and $t_{k+4}$ are updated.

when the voltage level of its located node is lowered, the voltage levels of all the computing nodes must be adaptively adjusted to ensure schedulability.

In Eq. (9), the start time $st_{ij}$ of task $t_i$ on node $n_j$ is a value anywhere between $t_i$'s earliest start time $est_{ij}$ and latest start time. In this paper, the start time $st_{ij}$ is measured by $t_i$'s earliest start time $est_{ij}$ to make $t_i$ finish as earlier as possible. Thus, it is important to compute the earliest start time as follows (see Eq. (11) for a formal description). If node $n_j$ is sitting idle, the earliest start time of task $t_i$ on node $n_j$ equals the task's arrival time. Otherwise, the task will have to wait in the local queue for all the tasks with earlier deadlines to be completed. Thus, $est_{ij}$ can be calculated as:

$$est_{ij} = a_i + \sum_{o_{kj} < o_{ij}, w_{kj}=1} e_{kj}(v_{kj}) + r_j, \quad (11)$$

where $r_j$ is the remaining execution time of the running task on node $n_j$.

**Property 2.** *The earliest start times of waiting tasks in a local queue will be recalculated if a higher voltage level is applied to accommodate a new task. The earliest start times should be updated in the following three cases:*

Case 1. *If any task $t_k$ is waiting in the local queue of node $n_j$, and $t_k$ has the earliest execution order, then*

$$est'_{kj} = est_{kj}. \quad (12)$$

Case 2. *If any task $t_k$ is waiting in the local queue of node $n_j$, and $t_k$'s execution order is earlier than that of new task $t_i$ but not the earliest execution order, then*

$$est'_{kj} = est_{kj} - \sum_{o_{kj} > o_{mj}, w_{mj}=1} e_{mj}(v_{mj}) - e_{mj}(v'_{mj}). \quad (13)$$

Case 3. *If any task $t_k$ is waiting in the local queue of node $n_j$, and $t_k$'s execution order is later than that of new task $t_i$, then*

$$est'_{kj} = est_{kj} + e_{ij}(v_{ij}) - \sum_{o_{kj} > o_{mj}, w_{mj}=1} e_{mj}(v_{mj}) - e_{mj}(v'_{mj}) \quad (14)$$

*where $est'_{kj}$ is the new earliest start time of task $t_k$ on node $n_j$, and $e_{mj}(v'_{mj})$ is the new execution time of task $t_m$ on node $n_j$ when a higher voltage level $v'_{mj}$ is applied to accommodate new task $t_i$.*

Fig. 2 illustrates a case described in the above Property 2.

If a cluster is under heavy load, it is likely that no computing node in the cluster can guarantee a new task's deadline. The reason is two-fold. First, tasks have longer execution time due to lower voltage levels on nodes. Second, the new task has later start time in one node, because many other tasks allocated to the same node are waiting in the local queue.

Our EEGS algorithm improves the adaptivity of the scheduling mechanism in the following way. If a new task cannot be allocated with current voltage levels of nodes in a cluster, then the voltage levels of some nodes will be increased to enhance the schedulability by admitting the new task rather than rejecting the task.

**Algorithm 1** Pseudocode of *EEGS* algorithm

```
1:  for each new task $t_i$ do
2:    $find \leftarrow FALSE$; $findNode \leftarrow NULL$; $v \leftarrow V_{max}$; $ec \leftarrow \infty$; $f \leftarrow \infty$;
3:    for each node $n_j$ in a cluster do
4:      Calculate the earliest start time $est_{ij}(v_{ij})$ using the highest voltage level, i.e., $v_{ij} \leftarrow V_{max}$;
5:      if $est_{ij}(v_{ij}) + e_{ij}(v_{ij}) \le d_i$ then
6:        while $v_{ij} \ge V_{min}$ do
7:          Degrade one voltage level of $n_j$: $v_{ij} - -$;
8:          Calculate the $est_{ij}(v_{ij})$ and finish time $f_{ij}$;
9:          if $est_{ij}(v_{ij}) + e_{ij}(v_{ij}) > d_i$ or $\exists t_k, o_{kj} > o_{ij}, w_{kj} == 1 : est_{kj}(v_{kj}) + e_{kj}(v_{kj}) > d_k$ then
10:           $v_{ij} + +$;
11:           break;
12:         end if
13:       end while
14:       Calculate the energy consumption $ec_{ij}$;
15:       if $(ec_{ij} < ec)$ or $(ec_{ij} == ec$ and $f_{ij} < f)$ then
16:         $v \leftarrow v_{ij}$; $findNode \leftarrow n_j$; $find \leftarrow TRUE$;
17:       end if
18:     end if
19:   end for
20:   if $find == TRUE$ then
21:     Allocate $t_i$ to $findNode$;
22:     Adjust the voltage level of $findNode$ to $v$;
23:   else
24:     Reject task $t_i$;
25:   end if
26: end for
```

EEGS is a heuristic scheduling algorithm. When a new task arrives, the admission test is performed on each node. Initially, the maximal voltage level is tested on a node, then EEGS decreases the voltage level of the node until the new task or waiting tasks cannot be completed before their deadlines. EEGS selects the node with the lowest energy consumption to save energy. Performing the maximal-voltage-level-admission test is to guarantee high schedulability. If all the nodes running at the highest voltage levels cannot accommodate the new task, then the task is rejected.

The pseudocode of EEGS is described in *Algorithm* 1. EEGS tests if a new task's deadline can be guaranteed with the highest voltage level of a node (see Lines 4–5). If the initial test is passed, EEGS decreases the voltage level of the node until the new task or waiting tasks in the node's local queue cannot have their timing constraints satisfied (see Lines 6–13). An iteration is performed on each node to obtain the lowest voltage level. After the energy consumption at the lowest voltage levels is computed, EEGS chooses a node with the minimum energy consumption. If several nodes have same minimum energy consumption, the node offering the earliest finish time to the new task is selected to break the tie (see Lines 14–17). If all nodes cannot admit the new task even at the highest voltage levels, the task is rejected. Otherwise the new task is allocated (see Lines 20–25).

The time complexity (see the Theorem below) of EEGS depends on the number of computing nodes in a cluster, the number of tasks in a local queue, and the number of voltage levels.

**Theorem 1.** *The time complexity of the EEGS scheduling algorithm is $O(mkq)$, where m is the number of computing nodes, q is the number of tasks in a local queue, and k is the number of voltage levels.*

**Proof.** The time complexity of calculating the earliest time of tasks in a local queue is $O(q)$ (see line 4 and line 8). It takes $O(k)$ (see lines 6–13) to determine the lowest voltage level (i.e., lowest energy consumption to accommodate a newly arrived task). It only takes $O(1)$ to execute the other lines. Hence, the time complexity of EEGS is calculated as follows: $O(m)(O(k) + O(k)(O(q))) = O(mkq)$. □

### 4.2. The LVA algorithm

The LVA algorithm is implemented in the local voltage adjuster to dynamically control voltages to conserve energy. LVA strives to decrease voltage level of a node after each running task finishes its execution. The design of LVA is indispensable, because a node's voltage may be scaled up since a running task needs a higher node voltage to guarantee the task's deadline. Tasks in the node's local queue may just need low voltage to satisfy their timing constraints. In our design, node voltages are scaled when all the tasks in a local queue are considered together to meet their timing requirements. Thus, all the temporary supply voltages of tasks in a local queue are degraded if the tasks' timing constraints can be guaranteed. When another task is finished, LVA is invoked again. Therefore, the actual supply voltages for the tasks may be different. LVA is able to efficiently reduce energy consumption and enhance the system adaptivity. The LVA has the following properties.

**Property 3.** *The voltage level of a node can be decreased and the following constraint must be met.*

$$\forall t_k, \quad w_{kj} = 1 : est'_{kj} + e_{kj}(v'_{kj}) \le d_k, \tag{15}$$

where $est'_{kj}$ and $e_{kj}(v'_{kj})$ denote the new earliest start time and new execution time after the voltage level of node $n_j$ is degraded.

Since the execution times of tasks in a local queue may be increased, some tasks can miss their deadlines due to either an increased execution time or a later start time. The start time of waiting tasks must be recomputed if a new task is admitted. Thus, we have the following property.

**Property 4.** *The earliest start time of waiting tasks in a local queue must be recalculated if the voltage level is decreased for energy saving purpose.*

$$est'_{kj} = est_{kj} + \sum_{o_{kj} > o_{mj}, w_{kj}=1, w_{mj}=1} (e_{mj}(v'_{mj}) - e_{mj}(v_{mj})). \tag{16}$$

The pseudocode of LVA is outlined in *Algorithm* 2.

**Algorithm 2** Pseudocode of *LVA* algorithm

```
1:  Get the voltage level $v_p$ of node $n_q$;
2:  while $v_p > V_{min}$ do
3:    Degrade one voltage level of node $n_q$: $v_p - -$;
4:    for each task $t_k$ waiting in local queue of $n_q$ do
5:      Calculate the new earliest start time $est'_{kq}$ and the new execution $e_{kq}(v'_p)$;
6:      if $est'_{kq} + e_{kq}(v'_p) > d_k$ then
7:        Enhance one voltage level of node $n_q$: $v_p + +$;
8:        break;
9:      end if
10:   end for
11: end while
```

The objective of LVA is to minimize the voltage levels of computing nodes. If the voltage level of a node is not set to the lowest value, LVA will test if a lower voltage level can also guarantee the deadlines of all the waiting tasks in the local queue.

If the deadlines can be met, LVA calculates the new earliest start times and execution times of the waiting tasks under the lower voltage level (see Line 5 in Algorithm 2). If the lower voltage level fails to guarantee the deadline of any task, the voltage-level degradation process is terminated and the voltage level is rolled back to the former value (see Lines 6–9 in Algorithm 2). The time complexity of LVA can be found in the following theorem.

**Theorem 2.** *The time complexity of the LVA algorithm is $O(kq)$, where $k$ is the number of voltage levels and $q$ is the number of waiting tasks in a local queue.*

**Proof.** It takes $O(q)$ time to check whether or not all the tasks in the local queue can meet their deadlines when a lower voltage level is chosen. The time complexity of adjusting voltage levels is $O(k)$. Therefore, the time complexity of LVA is $O(k)(O(q)) = O(kq)$. □

## 5. Performance evaluation

We evaluate in this section the performance of the proposed adaptive energy-efficient scheduling strategy (AEES). To demonstrate the performance improvements gained by AEES, we quantitatively compare it with an existing MEG (*m*inimum *e*nergy *g*reedy) algorithm presented in the literature [14,37], and two baseline algorithms—MEHV (the *m*inimum *e*nergy *h*ighest *v*oltage algorithm) and MELV (the *m*inimum *e*nergy *l*owest *v*oltage algorithm), in which static (highest or lowest) voltage levels are employed. Note that using baseline algorithms to demonstrate strengths of proposed algorithms is widely used in many similar studies (see for example, [13]).

MEG, MEHV, and MELV are briefly described as follows:

(1) *MEG*. For each task allocation, MEG selects a node that yields the least energy consumption by dynamically adjusting voltage levels.
(2) *MEHV*. All computing nodes run at the highest voltage level. A node offering the least energy consumption is chosen when a newly arrived task is allocated.
(3) *MELV*. All nodes run at the lowest voltage level. MELV selects a node providing the least energy consumption for each task allocation.

The performance metrics by which we evaluate the system performance include:

(1) Guarantee Ratio (*GR*) defined as: *GR* = Total number of tasks guaranteed to meet their deadlines/Total number of tasks ×100%.
(2) Total Energy Consumption (*TEC*) used to embody the total energy consumption.
(3) Energy Consumption Per Task (*ECPT*) calculated as: *ECPT* = *TEC*/Total number of tasks guaranteed to meet their deadlines.
(4) CPU Utilization (*CU*) expressed by: *CU* = Time to execute tasks/Total running time of nodes in a cluster.

Note that the values of *TEC* and *ECPT* are normalized; similar expressions can be found in the literature [18,20,13,16].

### 5.1. Simulation method and parameters

The frequencies and voltages used in our simulations were obtained by testing Athlon-64 machines (see Table 2). The processing speed at 2 GHz was assumed to be 10,000 MIPS in average [13]. To reflect node heterogeneities, we varied the CPU processing speed of Athlon-64 in the range between 4000 MIPS and 10,000 MIPS (see Table 3).

(1) The voltage levels of all the nodes in a simulated cluster are chosen from values in the range from 0.9 to 1.5 V with an increment of 0.1 V. Also, AEES can be used in a cluster where DVS intervals between nodes are different. To study the heterogeneity issue, we let parameter min MIPS represent processing speed in terms of MIPS (million instructions per second) with the lowest voltage. The processing speeds of the computing nodes under the lowest voltage is normally distributed in the range of minMIPS. Similarly, parameter maxMIPS represents the processing speed under the highest voltage level. The processing speed increases 1000 MIPS and frequency increases 0.2 GHz when the voltage level increases 0.1 V. Noted that if the lower bound (i.e., minMIPS) and upper bound (i.e., maxMIPS) increase, the node heterogeneity level is increased. Thus, a wide variety of heterogeneity levels can be adjusted by changing the values of minMIPS and maxMIPS.
(2) Parameter *taskSize* represents task size. Without loss of generality, we assume that the distribution of task size is a normal distribution. We examine three task sizes: small tasks, middle tasks and large tasks. For example, the size is 0.1–10 GI for small tasks, 10–40 GI for middle tasks, and 40–100 GI for large tasks. The tasks used in our experiments belong to synthetic tasks, which are flexible and rational to exhibit the task heterogeneity and to vary the system workload.
(3) The deadline assignments are controlled by the deadline base (Tbase) denoted as $\beta$, which determines if tasks have loose deadlines or tight deadlines. The deadline $d_i$ of task $t_i$ in Eq. (17) is chosen similar as that described in [29],

$$d_i = a_i + (1+\beta) \times e_i^{\max}, \tag{17}$$

where $e_i^{\max}$ is the maximal execution time computed below:

$$e_i^{\max} = \max\{e_{ij}(V_{j1})\}. \tag{18}$$

(4) The arrival rate of tasks abides Poisson distribution. *intervalTime* is a random positive real number to represent the timing interval between two consecutive tasks.

**Table 2**
Frequencies and voltages of simulated nodes in a cluster. The values are obtained by testing Athlon-64 machines.

| Frequency (GHz) | Voltage (V) | Average MIPS |
|---|---|---|
| 0.8 | 0.9 | 4,000 |
| 1.0 | 1.0 | 5,000 |
| 1.2 | 1.1 | 6,000 |
| 1.4 | 1.2 | 7,000 |
| 1.6 | 1.3 | 8,000 |
| 1.8 | 1.4 | 9,000 |
| 2.0 | 1.5 | 10,000 |

Table 3 summarizes the values of parameters used to simulate heterogeneous computing nodes and real-time tasks submitted to heterogeneous clusters. The task size parameter in Table 3 is measured in term of the sum of CPU instructions using the execution-time estimation techniques mentioned in Section 3.2.

The energy consumption of each computing node can be easily measured in our simulation studies. For example, assume parameter minMIPS is [2000, 6000], maxMIPS is [8000, 12 000], *taskSize* is [10, 60], $\alpha$ is 1, node number is 64, and task number is 2048, then, based on the operating points in Table 2, we can calculate the normalized energy consumption of the *i*th task on *j*th node at 1.2 V as follows: $en_{ij}(v_{ij}) = 1 \cdot 1.4 \cdot 1.2^2 \cdot (10 + (60 - 10)/2048 \cdot i) \cdot 10^3/(5000 + (9000 - 5000)/64 \cdot j)$. As such, we can obtain the energy consumption of the *i*th task on the *j*th node at other voltages. It is worth noting that the arrival sequences of tasks are random in our simulations.

A basic yet important rule used in our simulations is "Once Tuning One Parameter (OTOP)", which has been widely applied in many simulation experiments reported in the literature [20,29,44,40,14]. In each experiment, we change a single parameter while keeping the other parameters fixed. Tuning one parameter at a time allows us to clearly observe impacts of the parameter on performance and energy efficiency of clusters.

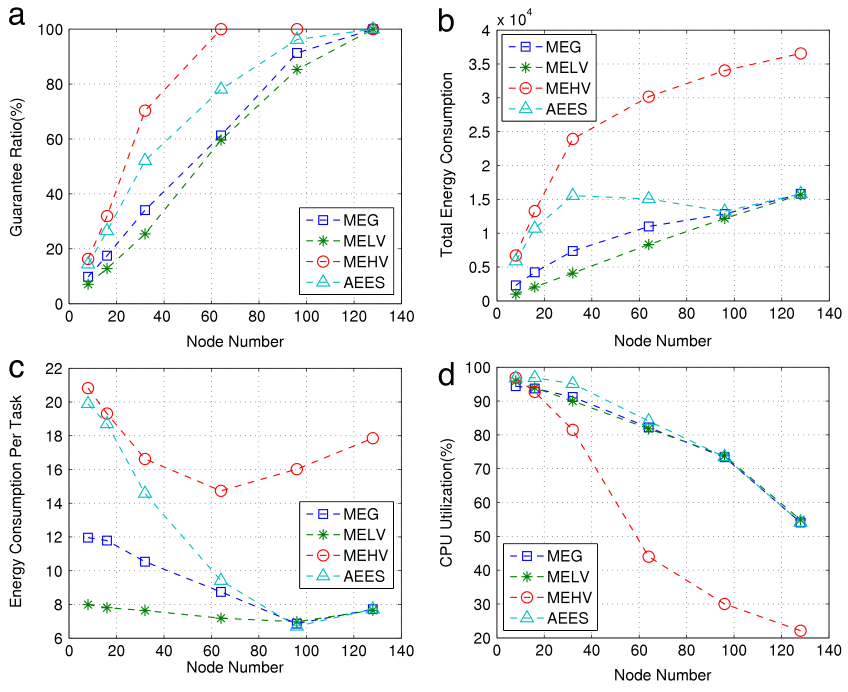

**Fig. 3.** Performance impact of node number.

**Table 3**
The values of parameters used to simulate heterogeneous computing nodes and real-time tasks submitted to heterogeneous clusters.

| Parameter | Value(fixed)–(varied) |
|---|---|
| Node number | (64)–(8, 16, 32, 64, 96, 128) |
| Task number | (2048) |
| minMIPS (MIPS) | ([2000, 6000])–([3000, 5000]), ([2000, 6000]), ([1000, 7000]) |
| maxMIPS (MIPS) | ([8000, 12 000])–([9000, 11 000]), ([8000, 12 000]), ([7000, 13 000]) |
| *intervalTime* (s) | (0.7)–(0.1, 0.3, 0.5, 0.7, 0.9, 1.1, 1.3, 1.5) |
| *taskSize* (GI) | ([10, 60])–([0.5, 10], [10, 60], [60, 100]) |
| *TBase* (s) | (2.0)–(1.1, 1.5, 2.0, 2.5, 3.0, 3.5) |

## 5.2. Scalability

In this experiment, let us study the scalability of our AEES strategy. The node number varies from 8 to 128. Fig. 3 shows the performances of MEG, MELV, MEHV, and AEES in terms of guarantee ratio, energy consumption, and CPU utilization.

We observe from Fig. 3(a) that MEHV always has higher guarantee ratio than those of the other schemes, because MEHV sets the voltage to the highest level all the time making tasks have less execution times and earlier start times. As a result, MEHV decreases the probability of missing deadlines. In contrast, MELV always keeps the lowest voltage level resulting in the lowest guarantee ratio. Besides, Fig. 3(a) shows that the guarantee ratio of AEES is higher than that of MEG. This result can be attributed to the fact that when the guarantee ratio does not achieve 100% reflecting heavy system workload, schedulability becomes the main objective of AEES, meaning that AEES strives to increase the voltage levels for waiting tasks in local queues. MEG simply increases the voltage level for the newly arrived tasks under heavy workload conditions, ignoring the waiting tasks in the local queues.

Although MEHV has the highest guarantee ratio (see Fig. 3(a) and (b)), MEHV's energy efficiency is the lowest regardless of the cluster's workload. MELV makes the cluster very energy efficient because of the lowest voltage levels. Both algorithms lack adaptivity, because neither MEHV nor MELV can adjust voltages according to the system load. Interestingly, Fig. 3(b) shows that the total energy consumption of the cluster when AEES is employed increases if the number of computing node is less than 32. The reason is that when the system is over loaded, AEES improves the schedulability by increasing voltages. When the number of nodes is larger than 32, AEES is able to efficiently enhance the energy efficiency of the cluster while yielding a high guarantee ratio. Further, we observe that when the number of nodes is larger than 96, the total energy consumption of the AEES-enabled cluster increases again. This is because when the number of nodes is further increased in the cluster, most of the computing nodes will be sitting idle. Many idle computing nodes can lead to an increased idle energy consumption even when most of the nodes run in the lowest voltage levels. Also, when the node number is less than 96, the energy consumption of the cluster powered by AEES is more than that of the same cluster supported by MEG. This is because when a real-time cluster is heavily loaded, our AEES firstly gives high priority to deal with schedulability at the cost of energy efficiency (see Fig. 3(a), AEES accepts more tasks than MEG does when the number of nodes is less than 96). Unfortunately, MEG is unable to offer high schedulability when the cluster's load is high.

Fig. 3(c) reveals that the energy consumption per task decreases first and then increases with the increase of the number of nodes. This is because when the number of nodes increases, the cluster's load becomes relatively light, which allows the tasks to be completed at lower voltage levels to conserve energy. On the other hand, when the number of nodes is large, a handful of computing nodes are sitting idle. Therefore, the energy consumption per task increases due to idle energy consumption caused by an increasing number of idle nodes. Fig. 3(c) shows that MEHV has the highest energy consumption per task; MELV has the least energy consumption per task. When the number of nodes is less than 96, a cluster under the AEES strategy consumes more energy than the same cluster under MEG. We attribute this trend to the fact that AEES adaptively adjusts voltage levels to improve schedulability when system load is heavy. When it comes to a large-scale cluster (e.g., the number of nodes is larger than 96), the energy efficiency of AEES is improved (i.e., AEES is as energy efficient as MEG). The experimental results demonstrate that AEES can adaptively decrease energy dissipation on heterogeneous clusters while achieving high schedulability.

Fig. 3(d) illustrates that AEES consistently outperforms the others in terms of the CPU utilization, meaning that AEES is able to efficiently utilize the CPU resources. MEHV has the worst CPU utilization, because MEHV uses the highest voltage yielding more idle time slots than the other schemes do. In contrast, AEES strives to lower voltage levels while guaranteeing tasks' deadlines. Thus, tasks scheduled by AEES have longer execution time with higher CPU utilization.

In this experiment, we set the number of tasks is a constant while varying the number of computing nodes. This experiment is equivalent to an experiment where the number of tasks is varied with a fixed number of nodes. Due to the space limit, we do not show the impact of the number of tasks on performance and energy efficiency. It is intuitive to show that the number of tasks can affect energy efficiency, because increasing the number of tasks can increase energy consumed by the tasks.

### 5.3. Workload conditions

To examine the performance sensitivities of the four algorithms or strategies to the arrival rate of tasks, in this set of experiments, we vary the parameter *intervalTime* from 0.1 to 1.5 with increment of 0.1. Fig. 4 plots the performance of MEG, MELV, MEHV, and AEES.

Fig. 4(a) shows that when the parameter *intervalTime* is small, the high arrival rate makes a large number of tasks wait in the local queues. Thus, some late arriving tasks may miss their deadlines. With the increase of *intervalTime*, the number of tasks waiting in local queues decreases and tasks have earlier start times. Such light workload gives rise to a high guarantee ratio. Again, MEHV and MELV have the highest and lowest guarantee ratios, respectively. The guarantee ratio of AEES is higher than that of MEG. This result is consistent with that plotted in Fig. 3(a).

Fig. 4(b) reveals that the MEHV-enabled cluster consumes the most energy and the MELV-enable cluster is the most energy efficient one. It should be noted that our AEES has unique features. For example, when the parameter *intervalTime* is less than 0.3 (i.e., the system is heavily loaded), AEES makes an effort to improve the schedulability of clusters by increasing voltage levels for waiting tasks in local queues, whereas MEG has no intention to adjust the voltages for real-time tasks in the local queues. Therefore, compared with MEG, AEES causes clusters to consume more energy. When tasks arrive rate is low (i.e., light system load), AEES dynamically degrades the voltage supply to reduce energy consumption, thus, we find that AEES has similar energy consumption with MEG. The total energy caused by the AEES-enabled cluster increases again when *intervalTime* is larger than 0.9, because idle time is increasing when the workload becomes very light. An increasing idle time contributes the slightly increased energy consumption.

Fig. 4(c) shows that MEHV and MELV have the highest and lowest energy consumption per task. Again, we observe that AEES has good adaptivity in the sense that AEES trades low energy efficiency for high schedulability under heavy load. When the cluster's load becomes lighter, AEES favors high energy efficiency against guarantee ratio. An interesting result plotted in Fig. 4(c) is that the four schemes lead to more energy consumption per task when the *intervalTime* is large enough. This is because with the increasing arrival rate, the cluster has longer active times. Thus, the total energy consumption increases. However, the accepted task number has little impact to make this phenomenon happen. AEES causes more energy than MEG when *intervalTime* is smaller than 0.9, which can be attributed to the high schedulability of AEES.

Fig. 4(d) shows that AEES has the best performance in terms of CPU utilization. This is especially true when task arrival rate is high. The results are consistent with those plotted in Fig. 3(d).

### 5.4. Deadlines

The goal of this experiment is to investigate the impacts of task deadlines on the performance of MEG, MELV, MEHV, and AEES. We vary the parameter *TBase* from 1.1 to 3.5. Fig. 5 plots performances of the four policies.

It is observed from Fig. 5(a) that with the increase of *TBase* (i.e., deadlines become looser), the guarantee ratios of MEG, MELV, MEHV, and AEES increase accordingly. This trend is true because the time constraints are not very tight and tasks can be finished at later timing instants. In addition, we observe from Fig. 5(a) that MEHV and MELV have the highest guarantee ratio and lowest guarantee ratio, respectively. AEES's guarantee ratio is higher than that of MEG. This result is consistent with the one observed from the previous experiments (see Figs 3(a) and 4(a)).

MEHV has the worst energy efficiency (see Fig. 5(b)), although MEHV's guarantee ratio is the highest (see Fig. 5(a)). Basically, all the schemes keep the similar total energy consumption regardless of the change of task deadlines. The reason is that the total execution time is increased due to the long queues of nodes after the last task arrives, but the voltages are degraded because of looser deadlines. Therefore, the total energy consumption remains unchanged. In addition, we observe that unlike MEG, AEES always makes clusters consume more energy consumption. This observation is reasonable under high workload conditions, because AEES tries to accept more real-time tasks at the cost of energy.

Results plotted in Fig. 5(a) and (b) can be used to explain the trend observed in Fig. 5(c). The total energy consumption does not dramatically change, but the number of accepted tasks increases when the deadlines of tasks become looser. As a result, the energy consumption per task decreases accordingly.

Fig. 5(d) shows that AEES has the similar performance in terms of CPU utilization with MEG, and is better than MEHV and MELV. The results indicate that AEES can efficiently use the CPU resources in the cluster while making best trade-off between energy efficiency and schedulability.

### 5.5. Task size

We evaluate in this set of experiments the performance impact of task size. Three tested configurations of task size can be found in Table 3. We assume that the distribution of the size is a normal distribution. Fig. 6 shows the performances of MEG, MELV, MEHV, and AEES under small, middle, and large task sizes.

It is easy to conclude from Fig. 6(a) that when the task size is small, all the tested strategies have the 100% guarantee ratios due to short execution times. When it comes to median and large tasks, MEHV always offers the highest guarantee ratio; MELV's guarantee ratio is lowest. This is because MEHV and MELV are static algorithms and; therefore, they cannot adjust voltages according to the cluster's load. AEES has higher guarantee ratio than MEG, because when the cluster is heavily loaded, AEES can improve the schedulability at the cost of energy efficiency.

Fig. 6(b) illustrates the cluster's total energy consumption when the task size is small. Under this light workload, MELV, MEG, and AEES have similar energy efficiency except MEHV. This means that AEES can conserve energy at light system load. However, when the tasks have medium or large size (i.e., heavy system load), AEES increases energy consumption to improve system schedulability. Although MEG makes clusters consume less energy than AEES does when task sizes are medium or large, MEG's schedulability is obviously inferior to that of AEES.

Energy consumption per task shown in Fig. 6(c) proves that the adaptivity of AEES is high. These results are consistent with the ones plotted in Figs. 4(c) and 5(c).

Fig. 6(d) proves that AEES has better CPU utilization than all the other scheduling strategies, indicating AEES outperforms the alternative solutions regardless of task size.

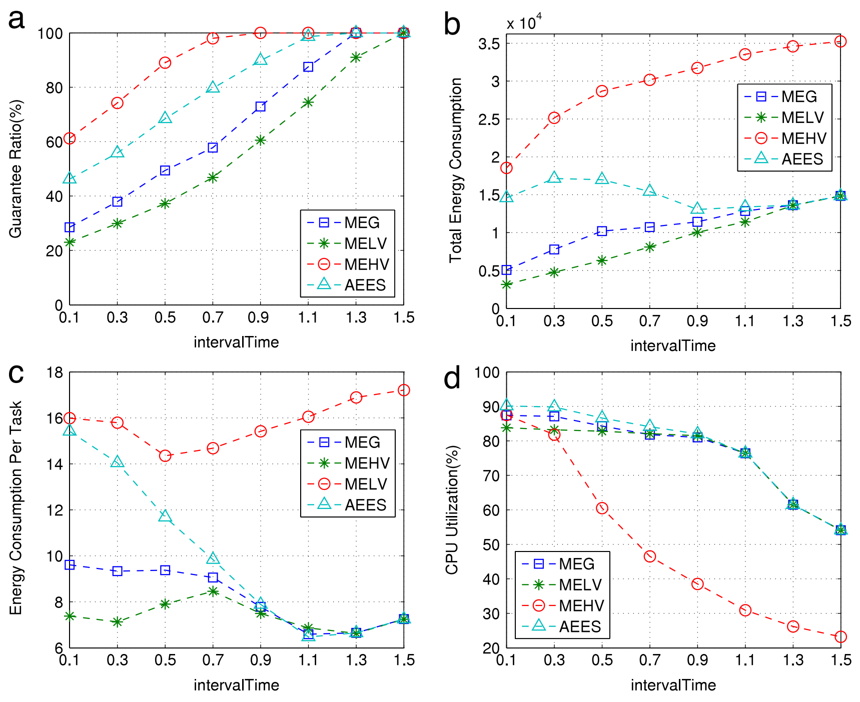

**Fig. 4.** Performance impact of arrival rate.

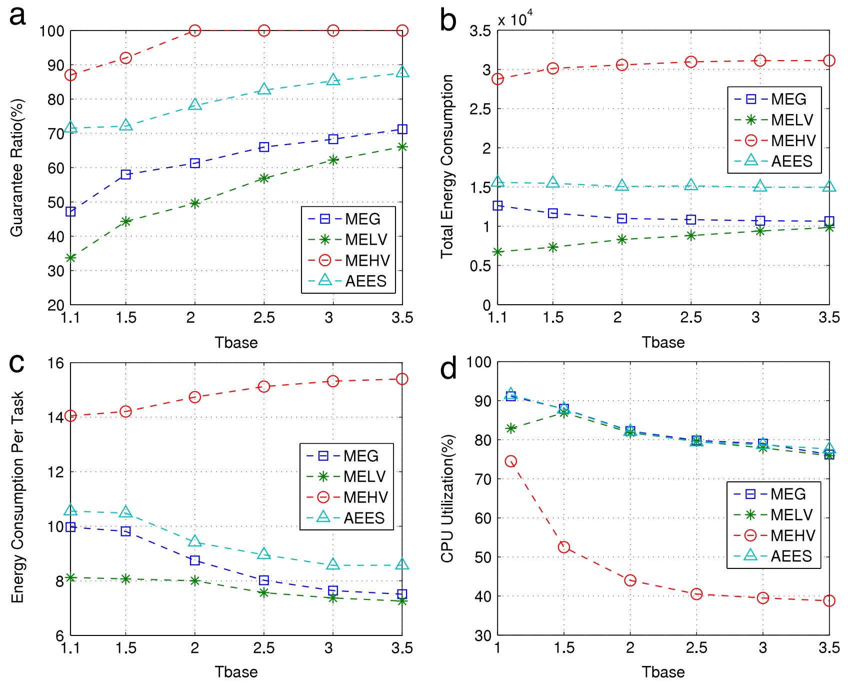

**Fig. 5.** Performance impact of task deadline.

### 5.6. *Heterogeneity levels*

In this experiment, let us investigate the impact of node heterogeneity on the cluster performance. Specifically, we evaluate three node heterogeneity degrees: small heterogeneity, middle heterogeneity, and large heterogeneity. Fig. 7 depicts the performances of MEG, MELV, MEHV, and AEES under three different heterogeneity levels.

Fig. 7(a) shows that the guarantee ratios of all the evaluated strategies are slightly enhanced with the increasing heterogeneity level. This can be attributed to the fact that the increased tasks accepted by nodes whose processing power increased exceed the decreased tasks accepted by nodes whose processing power decreased. Moreover, we realize that our AEES always has higher guarantee ratio than the other schemes except MEHV regardless of the heterogeneity level. This is because AEES can adjust the voltages of the new tasks as well as waiting tasks to improve

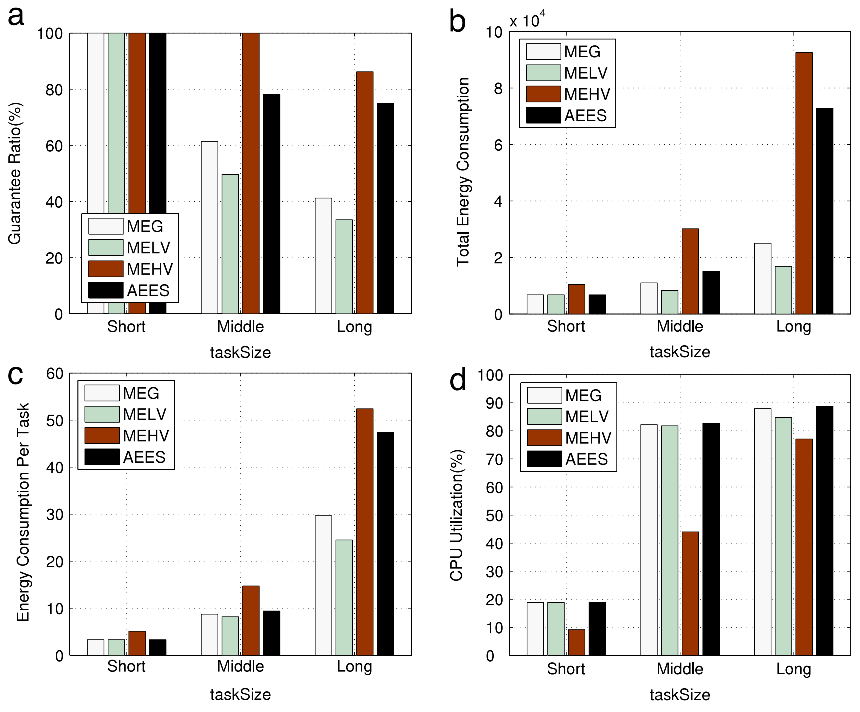

**Fig. 6.** Performance impact of task size.

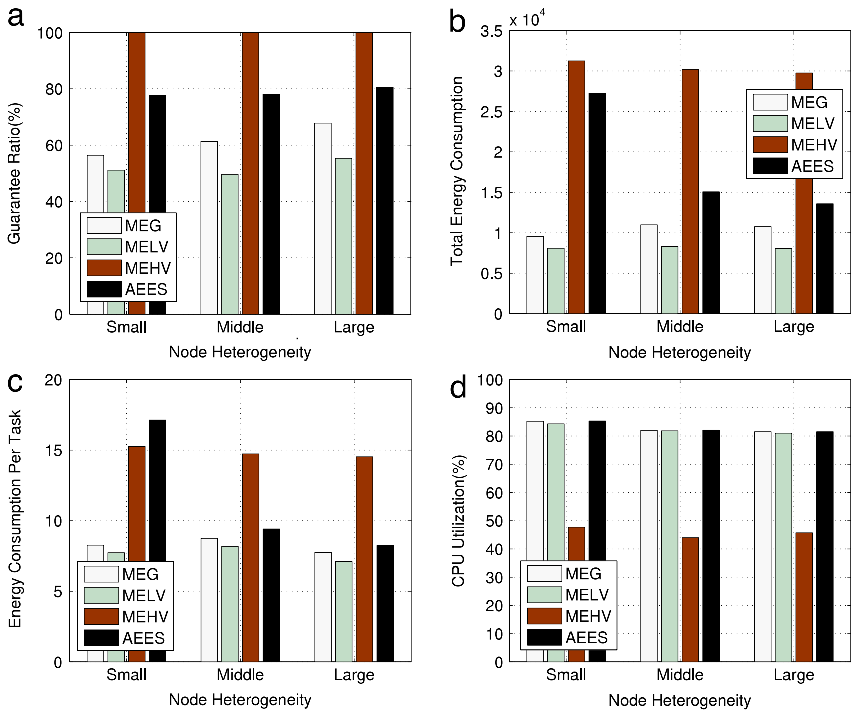

**Fig. 7.** Performance impact of node heterogeneity.

schedulability, whereas MEHV holds the highest guarantee ratio by consuming the most energy without any adaptivity.

Fig. 7(b) indicates that when the heterogeneity level is small, AEES leads to more total energy consumption in the cluster than MELV and MEG do. A low heterogeneity level means low processing performance and; thus, AEES strives to increase energy consumption to improve schedulability. However, when the heterogeneity level becomes high, the cluster has higher processing capability, AEES is able to reduce energy consumption as much as possible. Therefore, we can observe that the performance difference among AEES, MEG and MELV becomes smaller than that under low heterogeneity level. Additionally, MEG has better energy efficiency than AEES, but AEES offers high schedulability under heavy workload.

Fig. 7(c) shows when the cluster has lower processing power, AEES treats the schedulability as the primary scheduling goal. Thus, the energy consumption per task of AEES is the highest. However, when the processing power becomes high, AEES can efficiently decrease energy consumption to achieve the best energy efficiency compared with the other three schemes.

Fig. 7(d) shows that AEES has the best performance in terms of CPU utilization. The implication behind these results is that AEES can demonstrate its strength in heterogeneous clusters where its node heterogeneity varies.

## 6. Conclusions and future work

We presented in this paper a novel adaptive scheduling strategy or AEES for aperiodic, independent real-time tasks on DVS-enabled heterogeneous clusters. AEES seamlessly integrates two algorithms—EEGS and LVA. EEGS is implemented in the scheduler that is able to adaptively adjust voltages according to system load to guarantee deadlines of all waiting tasks in local queues. LVA, implemented in the local adjuster, can decrease voltage levels of waiting tasks to conserve energy when a task is scheduled and dispatched to a computing node. With EEGS and LVA in place, AEES efficiently improves the adaptivity and schedulability of real-time heterogeneous clusters. The extensive simulation studies using practical system parameters show that AEES is an excellent energy-efficient scheduling strategy designed for DVS-enabled heterogeneous clusters in dynamic environments.

The AEES algorithm and our simulation studies are the first step toward the development of energy-efficient real-time scheduling mechanisms for heterogeneous clusters. In one of our future studies, we plan to develop a prototype using a real-world heterogeneous cluster to test the effectiveness of our proposed energy-efficient scheduling algorithm.

With the prototype system in place, we will address the following five issues in our future studies: First, we will extend our AEES strategy to deal with other computing resources in addition to CPU. Other resources to be considered include memory, network bandwidth, and data storage systems. Second, we will implement a new scheduling mechanism in which communication and dispatching times are taken into account. Third, we plan to investigate energy-efficient and fault-tolerant scheduling schemes for real-time heterogeneous clusters. Fourth, we will consider a way of scheduling real-time tasks in a batch manner. Last, we will integrate the Ant Colony algorithm and the Particle Swarm algorithm with our AEES algorithm to further optimize our scheduling objectives.

## Acknowledgments

The authors are grateful to the anonymous referees for their insightful suggestions and comments. This research was supported by the National Natural Science Foundation of China under Grant No. 61104180, the National Basic Research Program of China under Grant No. 6136101, the National High-Tech Research and Development Plan of China under Grant No. 2008AA7070412, the US National Science Foundation under Grants CCF-0845257 (CAREER), CNS-0917137 (CSR), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS).

## References

[1] http://www.transmeta.com.
[2] R. Bianchini, R. Rajamony, Power and energy management for server systems, Computer 37 (11) (2004) 68–74.
[3] T.D. Braun, H.J. Siegal, N. Beck, et al. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems, in: Proc. 8th Heterogeneous Computing Workshop, HCW 1999, April 1999, pp. 15–29.
[4] H.Y. Chang, K.C. Huang, C.Y. Shen, S.C. Tcheng, C.Y. Chou, Parallel computation of a weather model in a cluster environment, J. Comput.-Aided Civ. Infrastruct. Eng. 16 (5) (2001) 365–373.
[5] http://www.arm.com.
[6] G. Donoho, Building a web service to provide real-time stock quotes, in: MCAD. Net, February 2004.
[7] W. Feng, Making a case for efficient supercomputing, ACM Queue 1 (7) (2003) 54–64.
[8] A. Gara, A. Blumrich, D. Chen, G.L.-T. Chiu, P. Coteus, M. Giampapa, R.A. Haring, P. Heidelberger, D. Hoenicke, G.V. Kopcsay, T.A. Liebsch, M. Ohmacht, B.D. Steinmacher-Burow, T. Takken, P. Vranas, Overview of the blue Gene/L system architecture, IBM J. Res. Dev. 49 (2–3) (2005) 195–212.
[9] O. González, H. Shrikumar, J.A. Stankovic, K. Ramamritham, Adaptive fault tolerance and graceful degradation under dynamic hard real-time scheduling, in: Proc. 18th IEEE Real-Time Systems Symp., RTSS 1997, December 1997, pp. 79–89.
[10] C.C. Han, K.G. Shin, J. Wu, A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults, IEEE Trans. Comput. 52 (3) (2003).
[11] L. Hu, H. Jin, X. Liao, X. Xiong, H. Liu, Magnet: a novel scheduling policy for power reduction in cluster with virtual machines, in: Proc. 2008 IEEE Int'l Conf. Cluster Computing, CLUSTER 2008, September 2008, pp. 13–22.
[12] A.A. Khokhar, V.K. Prasanna, M.E. Shaaban, C.L. Wang, Heterogeneous computing: challenges and opportunities, IEEE Comput. 26 (6) (1993) 18–27.
[13] K.H. Kim, R. Buyya, J. Kim, Power-aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters, in: Proc. 7th IEEE/ACM Int'l Symp. Cluster Computing and the Grid, CCGrid 2007, May 2007, pp. 541–548.
[14] J. Kim, H.J. Siegel, A.A. Maciejewski, R. Eigenmann, Dynamic resource management in energy constrained heterogeneous computing systems using voltage scaling, IEEE Trans. Parallel Distrib. Syst. 19 (11) (2008) 1445–1457.
[15] G. Klimeck, M. McAuley, R. Deen, F. Oyafuso, G. Yagi, E.M. DeJong, T.A. Cwik, Near real-time parallel image processing using cluster computers, in: Proc. 1st Int'l Conf. Space Mission Challenges for Information Technology, SMC-IT 2003, July 2003, pp. 13–16.
[16] R. Kotla, S. Ghiasi, T. Keller, F. Rawson, Scheduling processor voltage and frequency in server and cluster system, in: Proc. 19th Int'l Symp. Parallel and Distributed Processing, IPDPS 2005, April 2005, pp. 234–241.
[17] Y.-K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surv. 31 (4) (1999) 406–471.
[18] G. Laszewski, L. Wang, A.J. Younge, X. He, Power-aware scheduling of virtual machines in DVFS-enabled clusters, in: Proc. IEEE Int'l Conf. Cluster Computing, August 2009, pp. 1–10.
[19] W. Li, L. Kavi, R. Akl, A non-preemptive scheduling algorithm for soft real-time systems, J. Comput. Electr. Eng. 33 (1) (2007) 12–29.
[20] C. Liu, X. Qin, S. Li, PASS: power-aware scheduling of mixed applications with deadline constraints on clusters, in: Proc. 17th Int'l Conf. Computer Communications and Networks, ICCCN 2008, August 2008.
[21] http://www.dostor.com.
[22] G. Magklis, G. Semeraro, D. Albonesi, S. Dropsho, S. Dwarkadas, M. Scott, Dynamic frequency and voltage scaling for a multiple-clock-domain microprocessor, IEEE Micro 23 (6) (2003) 62–68.
[23] G. Manimaran, C.S.R. Murthy, A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis, IEEE Trans. Parallel Distrib. Syst. 9 (11) (1998) 1137–1152.
[24] M. Marinoni, G. Buttazzo, Adaptive DVS management through elastic scheduling, in: Proc. 10th IEEE Int'l Conf. Emerging Technologies and Factory Automation, ETFA 2005, September 2005, pp. 19–22.
[25] J. Markoff, S. Lohr, Intel's huge bet turns iffy, N. Y. Times Tech. Sec. Sec. 3 (2002) 1.
[26] H. Meuer, E. Strohmaier, J. Dongarra, H. Simon, TOP 500 supercomputing site. http://www.top500.org/, November 2009.
[27] M. Naedele, Fault-tolerant real-time scheduling under execution time constraints, in: Proc. 6th Int'l Conf. Real-Time Computing Systems and Applications, RTCSA 1999, December 1999, pp. 392–395.
[28] V. Nélis, J. Goossens, R. Devillers, D. Milojevic, N. Navet, Power-aware real-time scheduling upon identical multiprocessor platforms, in: Proc. 2008 IEEE Int'l Conf. Sensor Networks, Ubiquitous, and Trustworthy Computing, SUTC 2008, June 2008, pp. 209–216.
[29] X. Qin, H. Jiang, A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems, J. Parallel Comput. 32 (5) (2006) 331–356.
[30] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, Z. Lu, Power-aware QoS management in web servers, in: Proc. 24th IEEE Int'l Real-Time Systems Symp., RTSS 2003, pp. 63–72, December 2003.
[31] H.J. Siegel, H.G. Dietz, J.K. Antonio, Software support for heterogeneous computing, in: The Computer Science and Engineering Handbook, CRC Press, USA, 1997.
[32] J.D. Ullman, NP-complete scheduling problems, J. Comput. System Sci. 10 (3) (1975) 384–393.
[33] J. Wilkins, H.F. Sheikh, I. Ahmad, S.F. Khan, S. Rajput, Optimizing performance and energy in computational grids using non-cooperative game theory, in: Workshop in Work in Progress (WIPGC) in Conjunction with the 1st Int'l Conf. Green Computing, IGCC 2010, August 2010, pp. 343–355.
[34] T. Xie, X. Qin, Scheduling security-critical real-time applications on clusters, IEEE Trans. Comput. 55 (7) (2006) 864–879.

[35] T. Xie, X. Qin, M. Nijim, Solving energy-latency dilemma: task allocation for parallel applications in heterogeneous embedded systems, in: Proc. 2006 Int'l Conf. Parallel Processing, ICPP 2006, August 2006, pp. 12–22.
[36] M. Youssef, M. Younis, K. Arisha, A constrained shortest-path energy-aware routing algorithm for wireless sensor networks, in: Proc. IEEE Wireless Communication and Networks Conf., WCNC 2002, March 2002, pp. 129–136.
[37] Y. Yu, V.K. Prasanna, Power-aware resource allocation for independent tasks in heterogeneous real-time systems, in: Proc. 9th Int'l Conf. Parallel and Distributed Systems, ICPADS 2002, December 2002, pp. 341–348.
[38] K. Zheng, J. Wang, L. Huang, G. Decarreau, Open wireless software radio on common PC, in: Proc. 17th Ann. IEEE Int'l Symp. Personal, Indoor and Mobile Radio Communication, PIMRC 2006, September 2006, pp. 1–5.
[39] X. Zhu, P. Lu, A two-phase scheduling strategy for real-time applications with security requirements on heterogeneous clusters, J. Comput. Electr. Eng. 35 (2009) 980–993.
[40] X. Zhu, P. Lu, Multi-dimensional scheduling for real-time tasks on heterogeneous clusters, J. Comput. Sci. Tech. 24 (3) (2009) 434–446.
[41] D. Zhu, R. Melhem, B. Childers, Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems, IEEE Trans. Parallel Distrib. Syst. 14 (7) (2003) 686–700.
[42] X. Zhu, X. Qin, M. Qiu, QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters, IEEE Trans. Comput. 60 (5) (2011) 800–812.
[43] X. Zhu, J. Zhu, M. Ma, D. Qiu, SAQA: a self-adaptive QoS-aware scheduling algorithm for real-time tasks on heterogeneous clusters, in: Proc. 10th IEEE/ACM Int'l Conf. Cluster, Cloud and Grid Computing, CCGrid 2010, May 2010, pp. 224–232.
[44] Z. Zong, X. Qin, X. Ruan, K. Bellam, Energy-efficient scheduling for parallel applications running on heterogeneous clusters, in: Proc. 36th Int'l Conf. Parallel Processing, ICPP 2007, September 2007, pp. 19–26.

**Xiaomin Zhu** received the B.S. and M.S. degrees in computer science from Liaoning Technical University, Liaoning, China, in 2001 and 2004, respectively, and Ph.D. degree in computer science from Fudan University, Shanghai, China, in 2009. He is currently an assistant professor in the School of Information System and Management at National University of Defense Technology, Changsha, China. His research interests are green computing, cluster computing, fault-tolerant computing, and performance evaluation. He is a member of the IEEE, the IEEE Communication Society, and the ACM.

**Chuan He** received the B. S. and M. S. degrees in military operations research from Air Defense Forces Command Academy, Zhengzhou, China, in 2007 and 2010, respectively. He is currently a Ph. D. candidate in the School of Information System and Management at National University of Defense Technology, Changsha, China. His research interests include parallel and distributed computing, real-time systems and scheduling design.

**Kenli Li** received the Ph.D. degree in computer science from Huazhong University of Science and Technology, China, in 2003, and the M.S. degree in mathematics from Central South University, China, in 2000. He has been a visiting scholar at University of Illinois at Champaign and Urbana from 2004 to 2005. He is now a professor of Computer science and Technology at Hunan University. He is a senior member of CCF. His major research includes parallel computing, Grid and Cloud computing, and DNA computers.

**Xiao Qin** received the B.S. and M.S. degrees in computer science from Huazhong University of Science and Technology, Wuhan, China, in 1996 and 1999, respectively, and the Ph.D. degree in computer science from the University of Nebraska-Lincoln in 2004. He is currently an associate professor in the Department of Computer Science and Software Engineering at Auburn University. Prior to joining Auburn University in 2007, he had been an assistant professor with New Mexico Institute of Mining and Technology (New Mexico Tech) for three years. His research interests include parallel and distributed systems, real-time computing, storage systems, fault tolerance, and performance evaluation. He had served as a subject area editor of IEEE Distributed System Online (2000–2001). He has been on the program committees of various international conferences, including IEEE Cluster, IEEE IPCCC, and ICPP. He is a member of the IEEE and the IEEE Computer Society.