Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Slack allocation algorithm for energy minimization in cluster systems

Yikun Hu [a], Chubo Liu [a,b], Kenli Li [a,b,c,*], Xuedi Chen [a], Keqin Li [a,d]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China
[b] National Supercomputing Center in Changsha, Hunan, China
[c] CIC of HPC, National University of Defense Technology, Changsha 410073, China
[d] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

HIGHLIGHTS

- An energy-aware scheduling algorithm called EASLA is proposed.
- The main idea of the EASLA is to distribute slacks to tasks.
- The maximum set of independent tasks is involved.

ARTICLE INFO

Article history:
Received 12 March 2015
Received in revised form
11 July 2016
Accepted 31 August 2016
Available online xxxx

Keywords:
Cluster computing
Directed acyclic graph
Dynamic voltage/frequency scaling
Energy aware scheduling
Service level agreement

ABSTRACT

Energy consumption has been a critical issue in high-performance computing systems, such as clusters and data centers. An existing technique to reduce energy consumption of applications is dynamic voltage/frequency scaling (DVFS). In this paper, we present a novel algorithm called EASLA for energy aware scheduling of precedence-constrained applications in the context of Service Level Agreement (SLA) on DVFS-enabled cluster systems. Due to the dependencies among tasks and makespan extension, there may be some underused slacks. The main idea of the EASLA algorithm is to distribute each slack to a set of tasks and scale frequencies down to try to minimize energy consumption. Specifically, it first finds the maximum set of independent tasks for each task, and then iteratively allocates each slack to the maximum independent set whose total energy reduction is the maximal. Randomly generated graphs and two real-world applications are tested in our experiments. The experimental results show that our scheduling algorithm can save up to 22.68% and 12.01% energy consumption compared with the GreedyDVS and EvenlyDVS algorithms respectively in homogeneous environments, and 12.33% energy consumption compared with the EES algorithm in heterogeneous environments.

1. Introduction

With the development of high-performance computing (HPC), more and more energy has been consumed by large-scale cluster systems, which raises various economical, environmental and system availability concerns [1,2]. According to statistics, the total energy consumption in information and communication technology (ICT) industry is about 868 billion, which is roughly 5.3% of the world total energy consumption in 2008. According to the current increasing trend, by 2025, the total energy consumption in ICT industry will be four-fold increase on 2008 levels [3,4]. Additionally, energy consumption translates into high carbon emissions and results in high cooling cost [5]. Furthermore, the temperature of computing systems will ascend sharply due to large amount of energy consumption. Evidence shows that the expected failure rate doubles for every 10 °C increased temperature [6]. This greatly affects the system reliability and availability, and eventually does harm to system performance. Therefore, it is important to study the strategy of reducing energy consumption in high-performance computing.

Energy awareness for parallel applications has been a growing concern for a decade. System-level power-saving methods including Dynamic Power Management (DPM) [7] and Dynamic Voltage/Frequency Scaling (DVFS) [1,8,9] have been investigated and

* Corresponding author at: College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China.
E-mail addresses: yikunhu@hnu.edu.cn (Y. Hu), liuchubo@hnu.edu.cn (C. Liu), lkl@hnu.edu.cn (K. Li), chenxuedi@hnu.edu.cn (X. Chen), lik@newpaltz.edu (K. Li).

http://dx.doi.org/10.1016/j.future.2016.08.022
0167-739X/

developed. The core idea of DPM is to turn off the idle computing nodes. However, this method is only suitable for the situation in which most of the idle periods are very long. For the scheduling of a precedence-constrained application, the idle time between the execution of two tasks is often short and DPM is not suitable. Different from DPM, the DVFS technique can assign different frequencies to each task, which gives us a useful way to minimize energy consumption of applications.

In this paper, we use DVFS technique to reduce energy consumption while considering service level agreement (SLA), which is measured by two metrics: *makespan* and energy consumption. Obviously, there exists a tradeoff between these two metrics. In our architecture model, a customer can negotiate with a service provider about Quality of Service (QoS) of his application by determining *makespan* extension rate. For example, the customer agrees to accept additional 10% of *makespan* to reduce more energy consumption. Under this situation, we propose an energy-performance tradeoff scheduling algorithm (EASLA), which uses DVFS technique to try to minimize energy consumption. Our scheme addresses the scheduling in a different way compared with most of the relevant algorithms for discrete voltage clusters. First, it finds the maximum set of independent tasks for each task to increase parallelism for using slacks. Second, it also allocates slacks to non-critical tasks to minimize energy consumption instead of only the critical ones.

The main contributions of this paper are summarized as follows:

- Unlike the most existing studies for DAG applications, we try to allocate a slack to tasks belonging to the maximum independent set of a task.
- We develop a novel energy-aware scheduling algorithm called EASLA, which can be adapted for a wide range of DAG applications.
- We carry out extensive experiments to verify the effectiveness of our algorithm under different circumstances: randomly generated graphs as well as graphs of real-world problems with various characteristics, homogeneous resources in terms of speed and energy consumption, or heterogeneous resources.

2. Related work

Many works have been investigated for energy reduction based on continuous DVFS assumption. Zhang et al. introduced a linear programming (LP) based formulation to solve the continuous voltage DVFS [6]. Kang et al. developed a path based DVFS algorithm to try to minimize energy requirement while meeting the deadline constraints at the same time [10]. Aupy et al. proposed several polynomial time scheduling algorithms, which try to minimize energy consumption under the condition of a prescribed *makespan* bound and a reliability threshold [11]. Baskiyar et al. used the heterogeneous earliest finish time (HEFT) heuristic as an initial scheduling algorithm to minimize *makespan*, then voltage scaling was performed to reduce power consumption without performance degradation [12,13]. However, these algorithms focus on continuous frequency and voltage systems. For discrete DVFS consideration, the problem becomes more complex.

For the discrete frequency and voltage situation, the authors in [14–18,1,8,19] considered the scheduling and proposed some algorithms. Kimura et al. proposed an energy reduction algorithm, in which the suitable frequency among the discrete set of processor's frequencies is chosen for each task according to its slack time [14]. Rizvandi et al. introduced a slack reclamation algorithm which uses a linear combination of the maximum and minimum processor frequencies to decrease energy consumption [15]. Chowdhury et al. allocated the slack time to tasks according to the decreasing order of their finish times [16]. Wang et al. evenly distributed the free slack obtained by makespan extension to critical tasks, and then distributed the slack occurred due to dependencies among tasks to non-critical tasks [17]. The core idea of all these algorithms is to utilize idle time slots (slack) to lower down supply voltage (frequency/speed). However, most of them ignore the variable energy profiles of tasks on different processors during slack allocation or do not take into account the non-critical tasks for using idle time slots. Due to the dependencies among tasks, there may be many situations in which the sum of energy reduction of several tasks (i.e., a task set that can execute in parallel whether the tasks belonging to the set are critical or not) can be higher than the energy reduction of critical tasks. Our scheme addresses the above issues and effectively allocates the slack to gain more energy reduction for precedence-constrained applications in discrete frequency and voltage systems.

Many studies on cluster and cloud computing use DVFS technique in the context of Service Level Agreement (SLA) [20–25]. SLA is an agreement between a service provider and customers, specifying the level of delivered services [26,27]. It can be decided by many different QoS parameters, such as deadline [21], response time [22], and so on. Wu et al. proposed an algorithm to assign resources for tasks obeying the SLA [20]. In this algorithm, the tasks are deployed to fewer computing nodes with lower SLA level, and then appropriate frequencies are selected for the computing nodes via DVFS. Kim et al. proposed DVFS scheduling algorithms for bag-of-tasks applications to try to minimize energy consumption with deadline constraints [21]. Gao et al. proposed a dynamic resource management scheme which takes advantage of both DVFS and server consolidation to achieve energy efficiency while satisfying response-time-based SLAs [22]. In our work, we also consider the energy reduction problem in the context of SLA.

The remainder of this paper is organized as follows. In Section 3, we introduce the system model and formulate the scheduling problem. In Section 4, we describe the proposed EASLA algorithm and discuss its performance. Section 5 gives the simulation results compared with GreedyDVS and EvenlyDVS algorithms in homogeneous environments, and the EES algorithm in heterogeneous environments. Finally, Section 6 presents the conclusion and future work.

3. System model and problem formulation

In this section, we introduce the system model and formulate the scheduling problem.

3.1. Architecture model

In this subsection, we introduce the architecture model of our scheduling environment. As shown in Fig. 1, the architecture model, which is similar to [17], consists of three layers: user layer, scheduling layer, and resource layer. The user layer is in charge of submitting tasks. The scheduling layer, which is used to assign tasks appropriately, consists of two components—a *makespan* and energy estimator and our scheduling strategy. The resource layer consists of multiple processing elements (PEs) and is responsible for task execution. The needed PEs are allocated to users exclusively for some duration of time.

Next, we describe the task execution process in the scheduling model mentioned above. First, tasks with computation time and dependency relations, and the needed number of PEs are submitted by users. Second, the information of tasks and PEs are sent to the scheduler and it uses an estimator to evaluate the *makespan* and energy consumption. Third, the estimator gives the evaluated results back to the user. Fourth, the user negotiates with the service provider about the *makespan* extension rate, which presents the

Please cite this article in press as: Y. Hu, et al., Slack allocation algorithm for energy minimization in cluster systems, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.08.022

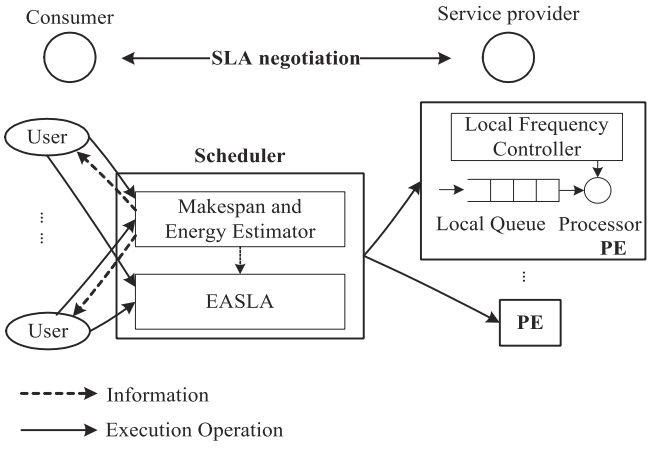


Fig. 1. Architecture model.

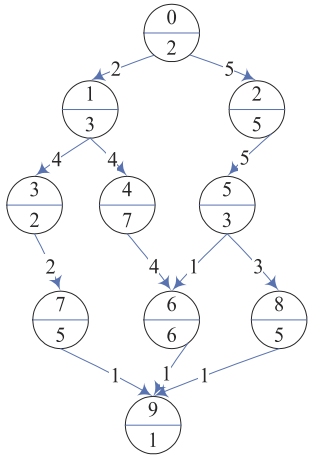


Fig. 2. A simple DAG representing precedence-constraint application.

QoS parameter. After an agreement is reached, the QoS parameter is delivered to the scheduler. Fifth, the scheduler allocates each task in the application with frequency and sequence information to an appropriate PE. Finally, the computing resources begin to execute tasks and return the results to users.

3.2. Application model

In general, a precedence-constrained application can be presented by a directed acyclic graph (DAG) $G(V, E)$, which consists of a set of nodes $V = \{v_i\}$ ($i \in \{0, 1, \ldots, n-1\}$) and a set of directed edges E. The set of nodes represents the parallel tasks in an application. An edge $e(i, j)$ from task v_i to v_j represents the dependency and inter communication between these two tasks. Here v_i is called a predecessor of v_j and v_j is called a successor of v_i. In this paper, we assume that the DAG has a single-entry task and a single-exit task. If there are multi-entry (multi-exit) tasks, they are connected to an entry (exit) task with no computation and communication cost, which does not affect the schedule.

The weight on task v_i denoted as w_i represents the computation time when v_i is executed at the highest frequency. Let c_{ij} be the weight of edge $e(i, j)$, which represents the communication time [17]. However, the communication time is only needed when two tasks are assigned to different PEs. In other words, the communication time is ignored when two tasks are assigned to the same PE. Fig. 2 gives an example of DAG to represent the application model. For each node, the data in the upper part presents the task ID and the data in the lower part presents the computation time. The data on each edge represents the communication time, i.e., c_{ij}.

3.3. Cluster model

Similar to [17], a cluster system consists of multiple PEs, which are fully interconnected with the same communication links. Each PE has a set of supply voltages from V_1 to V_m and a set of corresponding frequencies from f_1 to f_m in decreasing order. The values of voltage levels and corresponding frequencies are discrete. Table 1 shows an example of four voltage and frequency levels. As clock frequency transition time is negligible (e.g., 10–150 μs [1]), we ignore the time in our study. We also assume that data can be transmitted from one PE to another while the recipient PE is executing a task, which is possible in many computing systems.

3.4. Energy model

Our energy model is derived from the power consumption model in CMOS logic circuits. In general, the power consumption

Table 1
An example of a PEs operating points.

Frequency (GHz)	1.0	0.8	0.7	0.5
Voltage (V)	1.5	1.3	1.1	0.9

of CMOS circuits is represented by $P_c = ACV^2f$, where A is the percentage of active gates, C is the total capacitance load, V is the supply voltage, and f is the processor frequency [28]. Processor power consumption is dominated by P_c, which is mainly affected by the supply voltage. Therefore, the reduction of voltage would be most influential to lower power consumption. For a DAG application which consists of n tasks, the total energy consumed by executing all the tasks is defined as

$$EN_{active} = \sum_{i=0}^{n-1} P_{highest} \cdot w_i = \sum_{i=0}^{n-1} ACV^2_{highest} f_{highest} \cdot w_i, \quad (1)$$

where w_i is the computation time of task v_i executed at the highest frequency. Since PEs also consume a certain amount of energy while idling, we denote P_{idle} as the idle-energy consumption rate, where $P_{idle} = ACV^2_{lowest} f_{lowest}$. Denote p as the number of PEs, the total energy consumption of all the PEs while idling can be depicted as

$$EN_{idle} = P_{idle}\left(p \cdot makespan - \sum_{i=0}^{n-1} w_i\right). \quad (2)$$

Consequently, the total energy consumption of the application running on the PEs can be expressed as

$$EN = EN_{active} + EN_{idle}. \quad (3)$$

Note that this energy consumption model is compatible with DVFS technology. In our energy model, PEs have several voltage and frequency levels, and a scheduling algorithm may choose the appropriate voltage and frequency to save energy. For a task v_i executed at frequency f_k, we denote the voltage as V^i_k, the frequency as f^i_k, and the corresponding computation time as w^i_k, where $w^i_k = w_i f_{highest}/f^i_k$. In that case, $P_{highest}$ is replaced by P^i_k, where $P^i_k = AC(V^i_k)^2 f^i_k$. The corresponding computation time w_i is replaced with w^i_k.

3.5. Problem formulation

The energy-performance tradeoff scheduling problem stated before is formally defined as follows. Give an application consists of n parallel tasks, the required number of PEs p, the schedule

Please cite this article in press as: Y. Hu, et al., Slack allocation algorithm for energy minimization in cluster systems, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.08.022

length of a best effort schedule $makespan_{best}$, and the negotiated *makespan* extension rate η, allocate the slacks to the appropriate tasks for downscaling their frequencies to try to minimize energy consumption. More formally, the problem can be formulated as follows:

Minimize

$$\begin{aligned} EN &= EN_{active} + EN_{idle} \\ &= \sum_{i=0}^{n-1}\sum_{k=1}^{m} \beta_{ik} P_k^i w_k^i \\ &\quad + P_{idle}\left(p \cdot makespan - \sum_{i=0}^{n-1}\sum_{k=1}^{m} \beta_{ik} w_k^i\right), \end{aligned} \tag{4}$$

subject to

$$makespan \le makespan_{best} \cdot (1+\eta), \tag{5}$$

$$\sum_{k=1}^{m} \beta_{ik} = 1, \quad \beta_{ik} \in \{0, 1\}, \forall v_i \in V. \tag{6}$$

The timing constraint is enforced by (5). Since the available frequency range is finite and discrete, we use boolean variable β_{ik} as an indicator to reflect which frequency level k is used for task v_i. Assume we select a special f_j^i for task v_i in the above formulation, then the corresponding P_j^i and w_j^i are fixed. The question becomes to assign different frequencies to each task to try to minimize energy consumption without violating the timing constraint.

4. Algorithm design

In this section, we present the details of our algorithm EASLA. The algorithm takes the output schedule of ETF (Earliest Task First) algorithm as an initial input schedule. Our algorithm for energy minimization is an iterative approach that allocates a certain amount of slack to a task or a subset of tasks for frequency downscaling. In each iteration, a task (a subset of tasks) which can use its (their) slack to get the maximal energy reduction is (are) chosen to downscale frequency (frequencies). The EASLA algorithm actually contains two algorithms—frequency downscaling under the condition of not changing *makespan* algorithm, NCM for short, and frequency downscaling using accepted *makespan* extension algorithm, UAME for short. The former uses the slack which is caused by precedence relations among tasks while the latter uses the slack which is caused by *makespan* extension.

4.1. The outline of EASLA

The outline of EASLA is given in Algorithm 1. In the algorithm, Step 1 calls the NCM algorithm which is presented in Algorithm 2. In Algorithm 2, only the tasks which can use their slacks to scale down frequencies by at least one level are considered. Step 2 calls the UAME algorithm which is presented in Algorithm 3. In Algorithm 3, every task can potentially use the slack caused by *makespan* extension to scale down frequency. The execution of the NCM algorithm precedes the execution of the UAME algorithm to expect more energy saving by reducing the possibility of redundant slack allocation to the same tasks. Step 3 calls the NCM algorithm again, which is designed to take full advantage of slacks. Because there may be some slacks which are not large enough for even one task to scale down frequency by one level, we can integrate some small slacks to a large one and assign the new slack to an appropriate task.

Algorithm 1 : EASLA

Require:
The output schedule of the ETF algorithm;
$makespan_{best}$;
The negotiated *makespan* extension rate η.

Ensure:
A task schedule.

1: Call the NCM algorithm for frequency downscaling with $makespan_{best}$ unchanged
2: Call the UAME algorithm for frequency downscaling using accepted *makespan* extension
3: Call the NCM algorithm for frequency downscaling with extended *makespan* unchanged

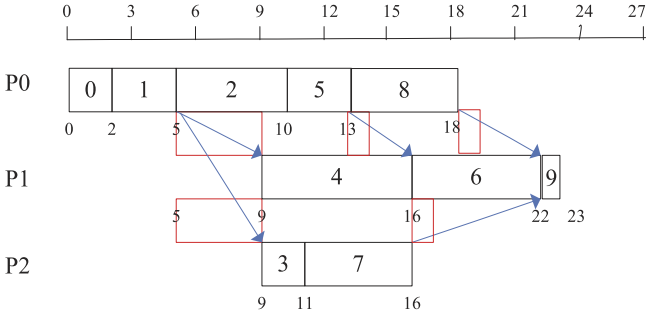


Fig. 3. A Gantt chart scheduled by the ETF algorithm.

4.2. Task scheduling algorithm

The ETF algorithm [29] is a list based algorithm, and has two main steps. It first assigns a priority value, which can be computed by the bottom level method or the top level method [30], to each task. Then the ETF algorithm repeats the process of allocating a remaining task with the highest priority to a PE, which has the earliest available time. The algorithm terminates when all of the tasks are assigned to the PEs. In this paper, we use the bottom level method, in which the *rank* value of each task is recursively defined by

$$rank(v_i) = w_i + \max_{v_j \in succ_i}(c_{ij} + rank(v_j)), \tag{7}$$

where $succ_i$ is the set of direct successors of task v_i and c_{ij} is the communication time between task v_i and v_j. The *rank* is computed by traversing the DAG upward, beginning from the exit task, v_{exit}, whose *rank* is expressed as follows:

$$rank(v_{exit}) = w_{exit}. \tag{8}$$

After a valid schedule is generated by the ETF algorithm and the order of tasks on a PE is determined, an edge is inserted from task v_i to task v_j if v_i and v_j are on the same PE and the former is scheduled just ahead of the latter [31]. Therefore, the edges represent all the precedence constraints in the original DAG application as well as execution ordering information in the initial schedule. We can use a Gantt chart to display time slots for task execution and data communication on multiple PEs. Fig. 3 gives an example of an initial schedule (Fig. 2 executed on three PEs) obtained by the ETF algorithm.

4.3. Available slack for a task

Notice that, the earliest start time of a task v_i is limited by the earliest finish time of the front adjacent task $pPred_i$ on the same PE,

Please cite this article in press as: Y. Hu, et al., Slack allocation algorithm for energy minimization in cluster systems, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.08.022

and the earliest finish times of all its direct predecessors. Therefore, the earliest start time of task v_i can be expressed as

$$EST_i = \begin{cases} 0, & v_i = v_{entry}; \\ \max\left(\begin{matrix} EST_{pPred_i} + w_{pPred_i}, \\ \max\limits_{v_j \in pred_i} \left(EST_j + w_j + c_{ji}\right) \end{matrix} \right), & \text{otherwise}; \end{cases} \quad (9)$$

where $pred_i$ is the set of direct predecessors of task v_i.

Similarly, the latest finish time of a task v_i is limited by the latest start time of the latter adjacent task $pSucc_i$ on the same PE, and the earliest start times of all its direct successors. Therefore, the latest finish time of task v_i can be expressed as

$$LFT_i = \begin{cases} makespan, & v_i = v_{exit}; \\ \min\left(\begin{matrix} LFT_{pSucc_i} - w_{pSucc_i}, \\ \min\limits_{v_j \in succ_i} \left(LFT_j - w_j - c_{ij}\right) \end{matrix} \right), & \text{otherwise}; \end{cases} \quad (10)$$

where *makespan* is the time overhead to complete all tasks.

For each task v_i, due to the difference between the latest finish time (LFT_i) and the earliest start time (EST_i), we can calculate the slack of task v_i as

$$slack_i = LFT_i - EST_i - w_i, \quad (11)$$

where w_i is the corresponding computation cost.

The energy reduction of task v_i is defined by the difference of its original energy consumption and expected energy consumption after allocating $slack_i$. However, slacks of some tasks are overlapped. So we can try to find tasks which can share slack together for each task. We use a matrix to present the relationship which is described in the next subsection.

4.4. Compatible task matrix

The matrix represents the list of tasks which can share slack together for each task or vice versa. The compatible matrix of a DAG application consisting of n tasks is defined by

$$M_n = \begin{bmatrix} m_{0\,0} & m_{0\,1} & \cdots & m_{0\,n-1} \\ m_{1\,0} & m_{1\,1} & \cdots & m_{1\,n-1} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ m_{n-1\,0} & m_{n-1\,1} & \cdots & m_{n-1\,n-1} \end{bmatrix}, \quad (12)$$

where m_{ij} indicates whether task v_i and task v_j can be slack-sharable. If the value of element m_{ij} is set to one, it indicates that v_i and v_j can share slack together because they are independently executed and their slacks are not overlapped. Otherwise, adding the slack to both of them will postpone the starting times of subsequent tasks and eventual violate the timing constraint. This corresponds to the problem of the reachability relation of a DAG. If v_j is reachable from v_i or v_i is reachable from v_j, it means that v_i and v_j cannot share slack together. The matrix can be generated by performing a transitive closure on the DAG [10]. Fig. 4 shows the compatible task matrix for the example in Fig. 3.

4.5. Slack allocation for frequency down-scaling

This subsection discusses slack allocation for frequency down-scaling.

Algorithm 2 represents the procedure of the NCM algorithm. The inputs of the algorithm are Γ and *makespan*. Γ represents a schedule of parallel tasks. In Step 1 of Algorithm 1, Γ is the schedule of the ETF algorithm and the corresponding *makespan* is $makespan_{best}$. In Step 3 of Algorithm 1, Γ is the schedule of Algorithm 3 and the corresponding *makespan* is the extended *makespan*. The task list denoted as L is initialized with all tasks in the application. The *EST*s (*LFT*s) of tasks are computed by Eq. (10)

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Fig. 4. Compatible task matrix for an example in Fig. 3.

Algorithm 2 : NCM(Γ, *makespan*)

Require:
Γ, *makespan*.
Ensure:
A task schedule.
1: Initialize $L = \{v_0, v_1, \ldots, v_{n-1}\}$
2: Sort all tasks such that $rank(v_0) \geq rank(v_1) \cdots \geq rank(v_{n-1})$ (*SRank*())
3: **while** $L \neq \varnothing$ **do**
4: **for** Each v_k in the order of *SRank*() **do**
5: Compute EST_k by Equation (10)
6: **end for**
7: **for** Each v_k in the reverse order of *SRank*() **do**
8: Compute LFT_k by Equation (11)
9: **end for**
10: Compute $slack_k$ ($v_k \in L$) by Equation (9)
11: **for** Each $v_k \in L$ **do**
12: **if** v_k can not be scaled down **then**
13: Delete v_k from L
14: **else**
15: Use $slack_k$ to compute ER_k
16: **end if**
17: **end for**
18: $v_j = \arg\max_{v_i} q(v_i)$ by Eq. (13)
19: Scale down $f^j_{current}$ and update $V^j_{current}$ and $w^j_{current}$
20: Delete v_j from L
21: **end while**

(11) in non-increasing (non-decreasing) order of *rank* denoted as *SRank*(), which can be done in time $O(n+e)$, where n is the number of tasks and e is the number of edges. Whenever a task is unable to scale frequency down, it is deleted from L. Denote $q(v_i)$ as the energy reduction of task v_i by using $slack_i$. In each iteration, the task v_j ($v_j \in L$) which can use its slack to get maximal energy reduction is chosen to scale frequency down, i.e., v_j is chosen as follows:

$$v_j = \arg\max_{v_i \in L} q(v_i). \quad (13)$$

Then the algorithm updates the frequency, voltage and computation time of v_j. The algorithm terminates when L becomes empty.

Algorithm 3 represents the procedure of the UAME algorithm. At the beginning of each iteration, it first finds a maximum set of independent tasks for each task (Step 1) [32]. We denote the maximum independent set for v_i as MIS_i. It can be used to maximize slack which can get more energy reduction. Even if the increased slack is not large enough for a task to scale down frequency by one level, it may be used at next iterations [33]. The number of tasks in MIS_i ($i \in \{0, 1, \ldots, n-1\}$) is restricted by the

Please cite this article in press as: Y. Hu, et al., Slack allocation algorithm for energy minimization in cluster systems, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.08.022

Algorithm 3 : UAME

Require:
The output schedule of NCM algorithm, time_extesion.
Ensure:
A task schedule.
1: *Find_MIS*()
2: Initialize $L = \{v_0, v_1, \ldots, v_{n-1}\}$, $ACT = \{0, 0, ..., 0\}$, $US = \{0, 0, ..., 0\}$
3: **while** $L \neq \varnothing$ **do**
4: **for** Each $v_k \in L$ **do**
5: **if** $f^k_{current} == f_1$ **then**
6: Delete v_k from L
7: **else**
8: $act_k = w^k_{current} \cdot \frac{f^k_{current}}{f^k_{current-1}} - w^k_{current}$
9: $slack_k = act_k - us_k$
10: **if** $slack_k > time\_extension$ **then**
11: Delete v_k from L
12: **else**
13: Use $slack_k$ to calculate OER_k
14: **end if**
15: **end if**
16: **end for**
17: $MIS_j = \arg\max_{MIS_i} u(MIS_i)$ by Eq. (15)
18: **for** Each $v_k \in MIS_j$ **do**
19: Scale down $f^k_{current}$
20: Update $V^k_{current}$, $w^k_{current}$, and us_k
21: **end for**
22: $time\_extension -= slack_j$
23: **end while**

number of PEs. The reason behind lies in the fact that there is at most one task on each PE that can be added to MIS_i. The compatible matrix can also provide the information of independence relations among tasks.

Denote tp_j ($i \in \{0, 1, \ldots, p-1\}$) as the set of tasks executed on processing element PE_j. Assuming that task v_i is on processing element PE_j, i.e., $v_i \in tp_j$. We try to find the maximum independent set for task v_i (MIS_i), which is initialized as an empty set. At first, we add task v_i to MIS_i, and then try to find the unique task $candidate_l$ on each of the other processing elements PE_l ($l \in \{0, 1, \ldots, p-1\}$, and $l \neq j$), which satisfies

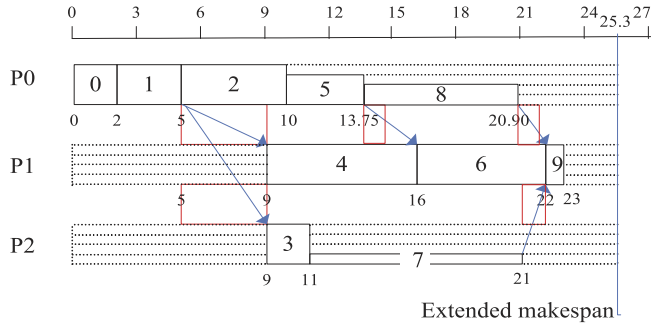
$$candidate_l = \arg\min_{v_k \in tp_l}(h(v_k)), \tag{14}$$

where $h(v_k) = EFT_i - EST_k$, $v_k \in tp_l$, and $EST_k \leq EFT_i$. If $candidate_l$ is slack-sharable with all the tasks in MIS_i, add it to MIS_i.

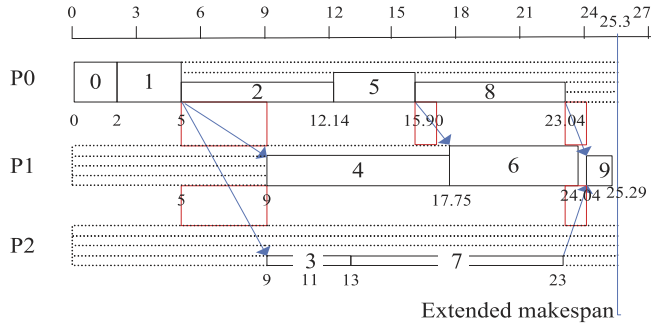
Algorithm 3 first initializes the task list with all the tasks in the application and the added computation time of tasks, i.e., $ACT = \{act_k\}$ ($i \in \{0, 1, \ldots, n-1\}$), and the unused slack, i.e., $US = \{us_k\}$ ($i \in \{0, 1, \ldots, n-1\}$), to zero. Whenever a task is unable to scale frequency down, it is deleted from L. Denote OER_i as the over energy reduction of all tasks in MIS_i to scale down frequency by one level when sharing $slack_i$. In each iteration, choose the task v_j ($v_j \in L$) with maximum value of OER_j to allocate its slack to all the tasks in MIS_i, i.e.,

$$v_j = \arg\max_{v_i \in L} u(MIS_i), \tag{15}$$

where $u(MIS_i) = OER_i$. Then the algorithm updates the frequency, voltage, computation time and unused slack of the tasks in MIS_j. The time extension is divided by the allocated slack $slack_j$ at the end of the iteration. The iteration stops when L becomes empty.



(a) The Gantt chart after Step 1 of Algorithm 1.



(b) The Gantt chart after Step 3 of Algorithm 1.

Fig. 5. A task schedule for Fig. 2 using the EASLA algorithm.

4.6. Performance evaluation

In this subsection, we discuss the qualitative implications of schedules that EASLA generates. The discussion is led with an illustrative example and the time complexity analysis.

4.6.1. An illustration

Consider the DAG application in Fig. 2 executed on three PEs. The information of the PEs is listed in Table 1. Fig. 3 gives the initial schedule obtained by ETF algorithm. At the beginning, all tasks are executed at the highest frequency. The $makespan_{best}$ is 23 and the total energy consumption is 99.9. The *makespan* extension is 2.3 (with *makespan* extension rate of 0.1). Fig. 4 shows the compatible task matrix for Fig. 3. In Step 1 of Algorithm 1, tasks 2, 3, 4, 7 and 8 are considered. Fig. 5(a) shows the Gantt chart after Step 1. The energy consumption is 82.80. The improvement of power consumption after Step 1 is, therefore, 17.29%. Fig. 5(b) shows the Gantt chart after Step 3 in Algorithm 1. The Gantt chart after Step 2 is the same with Fig. 5(b) except that the frequency of task 9 is f_1. The energy consumption is 70.35. Thus, the improvement on power consumption of our algorithm is 29.58% with *makespan* extension rate of 0.1.

4.6.2. Time complexity

Next, we discuss the time complexity of our algorithm EASLA (Algorithm 1).

According to Algorithm 2, Step 1 (Initialization) requires $O(n)$ to complete, where n is the number of tasks. Step 2 (Sorting) requires $O(n \log(n))$ to complete. Steps 3–21 are the iteration process. The upper bound of the iteration is $O(n)$. Because in each iteration, one task is chosen and this is a nonrepeatable scenario. In each iteration, Steps 4–6 and Steps 7–9 require $O(n + e)$ to complete, where e is the number of edges. The inner *for* loop (Steps 11–17) requires $O(n)$ to complete. Thus, the time complexity of NCM is $O(n(n + e) + n \log(n) + n)$.

Please cite this article in press as: Y. Hu, et al., Slack allocation algorithm for energy minimization in cluster systems, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.08.022

According to Algorithm 3, Step 1 is to find the maximum set of independent tasks for each task. The number of elements in the set is confined by the number of processor p, so Step 1 requires $O(np)$ to complete. Step 2 (Initialization) requires $O(n)$ to complete. The time complexity of iteration (Steps 3–23) is confined by the number of frequency levels m and the number of tasks n. Consider the situation that starting from f_1, all tasks are scaled down to f_m, then they have no chance to be scaled down. In each iteration, there will be at least one task to scale down frequency by at least one level. Hence, the loose upper bound of the time complexity is $O(n^2m)$. The time complexity of UAME is $O(n^2m + np + n)$.

Thus, the time complexity of EASLA is $O(n^2m + n(e + p + \log(n)) + n)$.

5. Extension to heterogeneous environment

In this section, we present the version of our proposed method applied on heterogeneous computing environment. We mainly present the different parts compared with homogeneous environment.

5.1. Differences on cluster model

Similar to cluster model in heterogeneous environment (Section 3.3), a cluster system consists of p heterogeneous PEs, which are fully connected with the same communication links. Denote the PE set as $\mathscr{S}_{PE}$, i.e., $\mathscr{S}_{PE} = \{1, \ldots, p\}$. Each PE l ($l \in \mathscr{S}_{PE}$) has a set of supply voltages, which is denoted as $\mathscr{V}_l = \{V_{l1}, \ldots, V_{lm_l}\}$, and a set of corresponding frequencies $\mathscr{F}_l = \{f_{l1}, \ldots, f_{lm_l}\}$, where m_l is the number of voltage (frequency) levels of PE l. The voltages and corresponding frequencies are discrete. We further denote the set of voltage (frequency) levels of PE l ($l \in \mathscr{S}_{PE}$) as $\mathscr{L}_{l,level} = \{1, \ldots, m_l\}$.

5.2. Differences on energy model

Our energy model is derived from the power consumption model in CMOS logic circuits. In general, the power consumption of CMOS circuits of PE l ($l \in \mathscr{S}_{PE}$) is represented by $P_l = A_l C_l V_l^2 f_l$, where A_l is the percentage of active gates, C_l is the total capacitance load, V_l is the supply voltage, and f_l is the processor frequency of PE l [28,18]. Processor power consumption is dominated by the power consumption of CMOS circuits, which is mainly affected by the supply voltage. Therefore, the reduction of voltage would be most influential to lower power consumption. There are n tasks and let $\mathscr{N}$ be the task set, i.e., $\mathscr{N} = \{1, \ldots, n\}$. Denote Φ_l ($\Phi_l \subseteq \mathscr{N}$) as the set of tasks on PE l ($l \in \mathscr{S}_{PE}$), then the total energy consumed by executing all the tasks on PE l is defined as

$$\begin{aligned} EN_{l,active} &= \sum_{i \in \Phi_l} P_{l,highest} \cdot w_i \\ &= \sum_{i \in \Phi_l} A_l C_l V_{l,highest}^2 f_{l,highest} \cdot w_i, \end{aligned} \tag{16}$$

where w_i is the computation time of task i executed at the highest frequency, and we obtain the total energy by executing all the n tasks as

$$\begin{aligned} EN_{active} &= \sum_{l \in \mathscr{S}_{PE}} EN_{l,active} \\ &= \sum_{l \in \mathscr{S}_{PE}} \sum_{i \in \Phi_l} A_l C_l V_{l,highest}^2 f_{l,highest} \cdot w_i. \end{aligned} \tag{17}$$

Since PEs also consume a certain amount of energy while idling, we obtain the power consumption of PE l ($l \in \mathscr{S}_{PE}$) on idle state as $P_{l,idle} = A_l C_l V_{l,lowest}^2 f_{l,lowest}$. The total energy consumption of all the PEs while idling can be depicted as

$$EN_{idle} = \sum_{l \in \mathscr{S}_{PE}} P_{l,idle} \left(makespan - \sum_{i \in \Phi_l} w_i \right). \tag{18}$$

Consequently, the total energy consumption of the application running on the PEs can be expressed as

$$EN = EN_{active} + EN_{idle}. \tag{19}$$

Notice that this energy consumption model is compatible with DVFS technology. In our energy model, each PE has several voltage and frequency levels. A scheduling algorithm is to choose the appropriate voltage and frequency to save energy. Note that after an initial allocation, i.e., allocating tasks on PEs, we do not move a task among PEs. We just try to take advantages of slacks among the tasks. For a task i ($i \in \mathscr{N}$) executed at PE l ($l \in \mathscr{S}_{PE}$) with frequency $f_{l,k}$, denote the corresponding computation time as w_k^i, where $w_{l,k}^i = w_i f_{l,highest} / f_{l,k}$. In that case, $P_{l,highest}$ is replaced by $P_{l,k}$, where $P_{l,k} = A_l C_l V_{l,k}^2 f_{l,k}$. The corresponding computation time w_i is replaced with $w_{l,k}^i$.

5.3. Differences on problem formulation

The energy-performance tradeoff scheduling problem stated before is formally defined as follows. Given an application consists of n parallel tasks, the required number of PEs p, the schedule length of a best effort schedule $makespan_{best}$, and the negotiated $makespan$ extension rate η, after an initial schedule $\mathbf{\Phi} = (\Phi_l)_{l \in \mathscr{S}_{PE}}$, allocate the slacks to the appropriate tasks for downscaling their frequencies to try to minimize energy consumption. More formally, the problem can be formulated as follows:

$$\begin{aligned} \text{minimize } EN &= EN_{active} + EN_{idle} \\ &= \sum_{l \in \mathscr{S}_{PE}} \sum_{i \in \Phi_l} \sum_{k \in \mathscr{L}_{l,level}} \beta_k^i P_{l,k} w_{l,k}^i \\ &\quad + \sum_{l \in \mathscr{S}_{PE}} P_{l,idle} \left(makespan - \sum_{i \in \Phi_l} \sum_{k \in \mathscr{L}_{l,level}} \beta_k^i w_{l,k}^i \right), \end{aligned} \tag{20}$$

$$\text{s.t. } makespan \le makespan_{best} \cdot (1 + \eta), \tag{21}$$

$$\sum_{k \in \Phi_l} \beta_k^i = 1, \quad \beta_k^i \in \{0, 1\}, \ \forall l \in \mathscr{S}_{PE}, \ \forall i \in \Phi_l. \tag{22}$$

The timing constraint is enforced by (21). Since the available frequency range after an initial schedule is finite and discrete, we use boolean β_k^i as an indicator to reflect which frequency level k is used for task i on the allocated PE l. The question becomes to assign different frequencies to each task to try to minimize energy consumption without violating the time constraint.

5.4. Same algorithm application

After the previous derivations on heterogeneous computing environment. We can apply the proposed EASLA algorithm for homogeneous situation on the heterogeneous environment, because the EASLA algorithm is unrelated to specific derivations.

6. Simulation and results

This section presents the simulation results obtained from our EASLA scheme. We compare the EASLA algorithm with two other

Please cite this article in press as: Y. Hu, et al., Slack allocation algorithm for energy minimization in cluster systems, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.08.022

Table 2
Voltage-relative speed pairs.

Level	Pair1		Pair2		Pair3	
	Voltage (V)	Frequency (GHz)	Voltage (V)	Frequency (GHz)	Voltage (V)	Frequency (GHz)
1	1.50	2.0	1.20	1.8	1.484	1.4
2	1.40	1.8	1.15	1.6	1.436	1.2
3	1.30	1.6	1.10	1.4	1.308	1.0
4	1.20	1.4	1.05	1.2	1.180	0.8
5	1.10	1.2	1.00	1.0	0.956	0.6
6	1.00	1.0	0.90	0.8		
7	0.90	0.8				

algorithms—EvenlyDVS [17] and GreedyDVS [16] in homogeneous environments, and the EES algorithm [18] in heterogeneous environments. In this work, we implement all these algorithms by C language and execute them on simulation environments to obtain the results. The EvenlyDVS algorithm evenly distributes the free slack obtained by *makespan* extension to critical tasks, and then distributes the slack occurred due to dependencies among tasks to non-critical tasks. GreedyDVS allocates the slack to tasks according to the decreasing order of their finish times. Both of them are designed for homogeneous environments. The EES algorithm evenly distributes the slack time between the *makespan* and the deadline to every task, and then tries to schedule the tasks nearby running on a uniform frequency for global optimality, which is designed for heterogeneous environments.

To test the performance of the scheduling algorithms, we simulate a cluster system with DVFS-enabled PEs. Three real heterogeneous PEs are chosen for our study and their voltage–frequency pairs are presented in Table 2. The PEs we choose in sequence are AMD Athlon-64, AMD Turion MT-34 [17] and Pentium M.

The performance is measured in terms of normalized total energy saving. Here, we define a parameter energy-saving-ratio *ESR* as the energy saving metric:

$$ESR = \frac{E_{ETF} - E}{E_{ETF}}, \tag{23}$$

where E_{ETF} is the energy consumption of the ETF algorithm with all tasks executed at the highest frequency, and E is the energy consumption of a compared algorithm with DVFS scheme. The *makespan* extension is determined by: $makespan \leq (1 + \eta) \cdot makespan_{best}$, where $makespan_{best}$ is the schedule length of a best effort ETF schedule. We provide experimental results for *makespan* extension rates equal to 0.0 (no *makespan* extension), 0.05, 0.1, 0.2, 0.3, and 0.4.

We consider two sets of graphs as the workflow applications for testing the algorithms: randomly generated application graphs and graphs that represent real-world applications.

6.1. Randomly generated application graphs

For the generation of random graphs, which are commonly used to compare scheduling algorithms, three fundamental characteristics of the DAG are considered: DAG size, n, communication to computation cost ratio, *CCR*, and parallelism factor, λ [7,30,34].

In our experiments, the method of generating random graphs is the same as [7]. Graphs were generated for all combinations of the above parameters. The number of nodes in DAG ranges from 100 to 800. To generate a DAG of a given number, the number of levels is determined by the parallelism factor λ(0.5, 1.0, 2.0, 5.0) firstly, and then the number of tasks at each level is determined. Edges are only generated between the nodes in the adjacent levels, obeying 0-1 distribution. The computation costs are taken randomly from a uniform distribution [10, 60] around 30. Thus the average computation cost is 30. The communication costs are also randomly selected from a uniform distribution, whose mean depends on *CCR*(0.2, 0.5, 1.0, 2.0) and the average computation cost. Every set of the above parameters is used to generate 100 random graphs in order to avoid scattering effects and then the average result is computed.

6.2. Performance results in homogeneous environments

In homogeneous environments, each PE is modeled with AMD Athlon-64. The information of AMD Athlon-64 is shown in Table 2.

6.2.1. Random application performance results

The first set of experiments compares the energy savings of algorithms with respect to various graph sizes (see Fig. 6). The performance on energy saving of EASLA outperforms EvenlyDVS and GreedyDVS for various numbers of tasks. For instance, when the *makespan* extension rate is 0.4, the *ESR* of EASLA algorithm is greater than the EvenlyDVS and GreedyDVS algorithms by: (9.97%, 18.06%), (9.53%, 16.55%), (9.97%, 15.14%), and (10.35%, 14.16%), for number of tasks of 100, 200, 400 and 800, respectively. We can observe that the energy savings of all the three algorithms tend to increase with the increase of *makespan* extension rate. However, the results of our proposed EASLA algorithm always outperform those of the others, which show the efficiency of the EASLA algorithm. The reason behind lies in that in our proposed algorithm, we try to allocate a slack to the tasks belonging to a maximum independent set, in which all the tasks can share the slack without overlaps.

The second set of experiments compares the energy savings of algorithms with respect to different number of PEs (see Fig. 7). When the *makespan* extension rate is 0.4, the *ESR* of EASLA algorithm is greater than the EvenlyDVS and GreedyDVS algorithms by: (9.97%, 15.14%), (8.55%, 17.65%), (8.03%, 19.54%), and (7.83%, 17.48%), for number of PEs of 10, 20, 30 and 40, respectively. We can also observe that the energy saving increases with the increase of the number of PEs, even though there is no makespan extension. The reason behind lies in the fact that when the number of PEs increases, there may be less tasks allocated on each PE. This implies that the earliest start time of a task takes less chances to be limited by the finish time of the front adjacent task on the same PE. Namely, a task can start in earlier time.

In the third group of experiments, we compare the energy savings of algorithms with respect to various *CCR*s. Table 3 shows the *ESR*s of the EASLA, EvenlyDVS and GreedyDVS algorithms for various *CCR*s at different *makespan* extension rates. When the *makespan* extension rate is 0.4, the *ESR* of EASLA algorithm is greater than the EvenlyDVS and GreedyDVS algorithms by: (9.97%, 15.14%), (9.61%, 14.04%), (7.86%, 12.09%), and (8.86%, 11.77%), for *CCR* of 0.2, 0.5, 1 and 2 respectively.

In the fourth group of experiments, we compare the energy savings of algorithms with respect to various parallelism degrees (see Fig. 8). When the *makespan* extension rate is 0.4, the *ESR* of EASLA algorithm is greater than the EvenlyDVS and GreedyDVS

Please cite this article in press as: Y. Hu, et al., Slack allocation algorithm for energy minimization in cluster systems, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.08.022

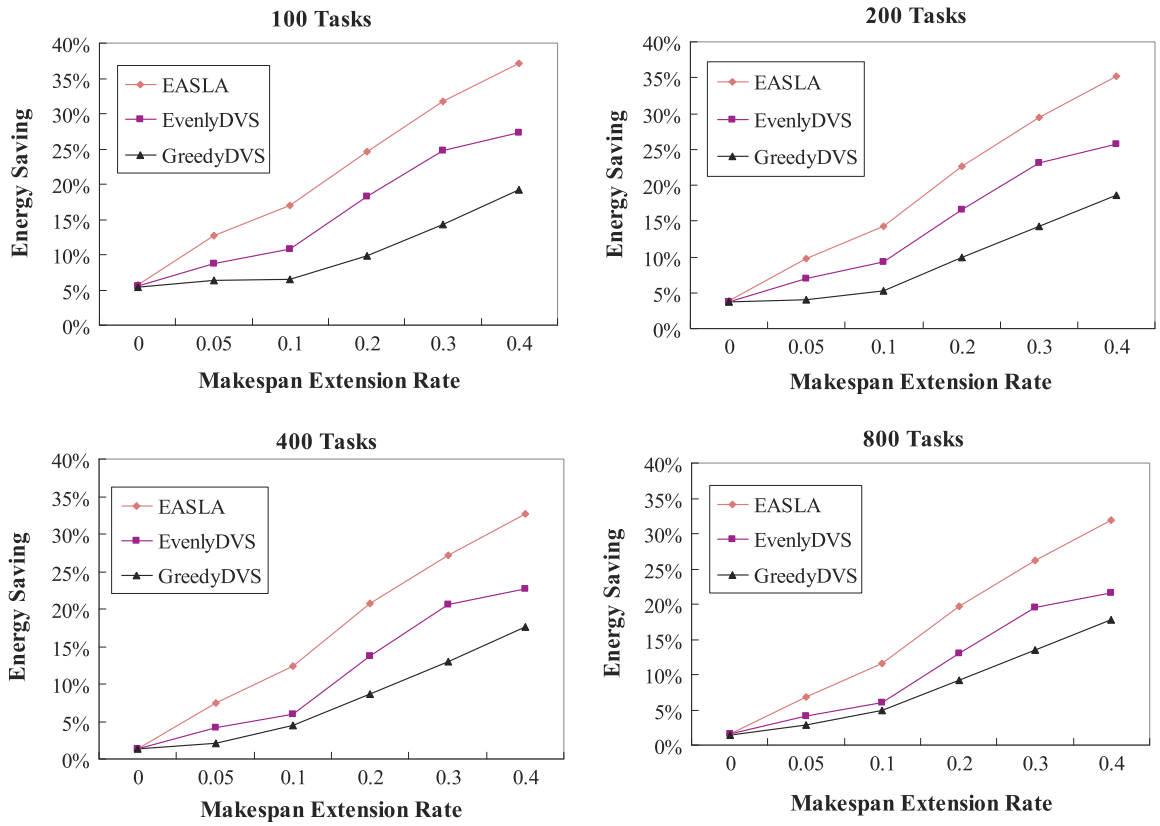


Fig. 6. Energy saving with respect to various numbers of tasks (10 PEs, $CCR = 0.2$, PARAFAC $= 2$).

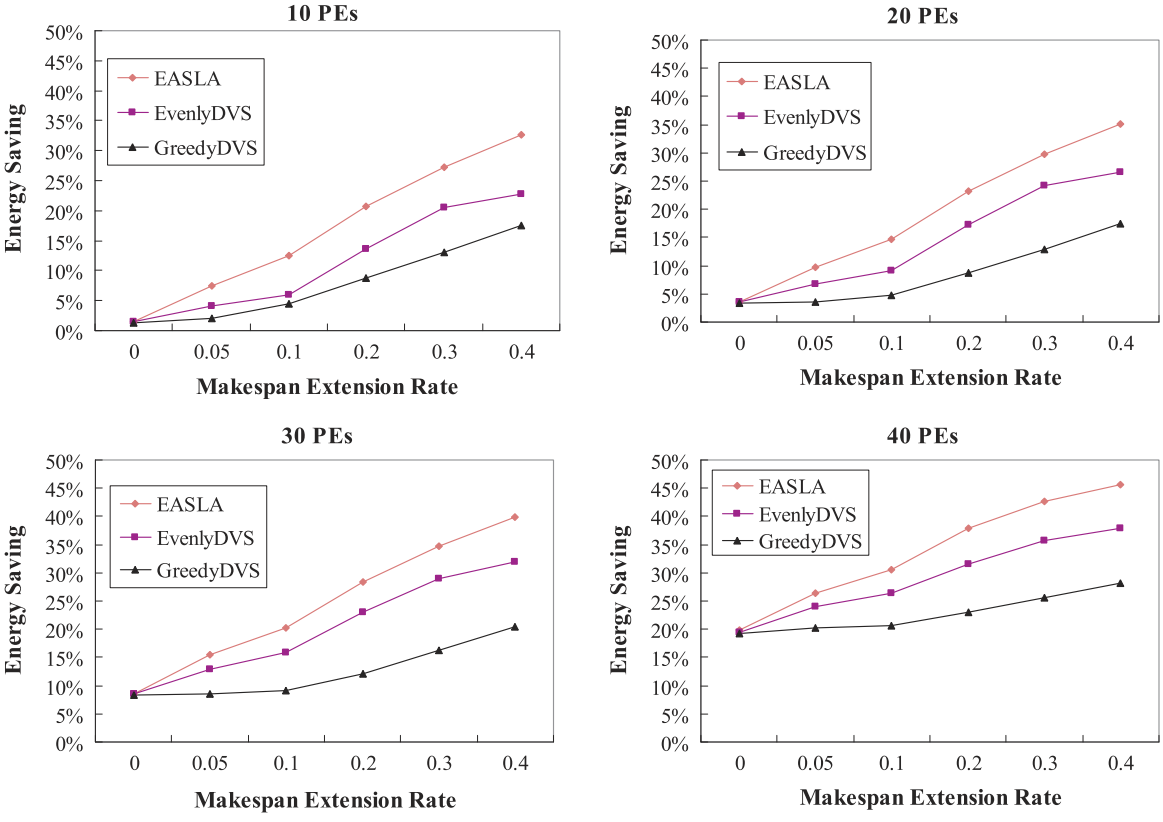


Fig. 7. Energy saving with respect to various numbers of PEs (400 tasks, $CCR = 0.2$, PARAFAC $= 2$).

Please cite this article in press as: Y. Hu, et al., Slack allocation algorithm for energy minimization in cluster systems, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.08.022

Table 3
ESRs of EASLA, EvenlyDVS and GreedyDVS for various *CCRs* at different *makespan* extension rates (400 tasks, 10 PEs, PARAFAC = 2).

CCR	0.2	0.5	1	2	0.2	0.5	1	2
	Makespan extension rate = 0.1				*Makespan* extension rate = 0.2			
EASLA	12.44%	11.94%	14.52%	18.68%	20.72%	20.93%	22.10%	27.54%
EvenlyDVS	6.03%	7.93%	11.59%	15.43%	13.70%	14.97%	18.11%	22.12%
GreedyDVS	4.42%	5.61%	9.01%	13.05%	8.70%	10.22%	13.93%	18.66%
	Makespan extension rate = 0.3				*Makespan* extension rate = 0.4			
EASLA	27.16%	27.85%	29.10%	34.42%	32.69%	33.74%	35.62%	41.27%
EvenlyDVS	20.61%	21.33%	24.08%	28.30%	22.72%	24.13%	27.76%	32.61%
GreedyDVS	13.04%	14.28%	18.19%	23.64%	17.55%	19.70%	23.33%	29.44%

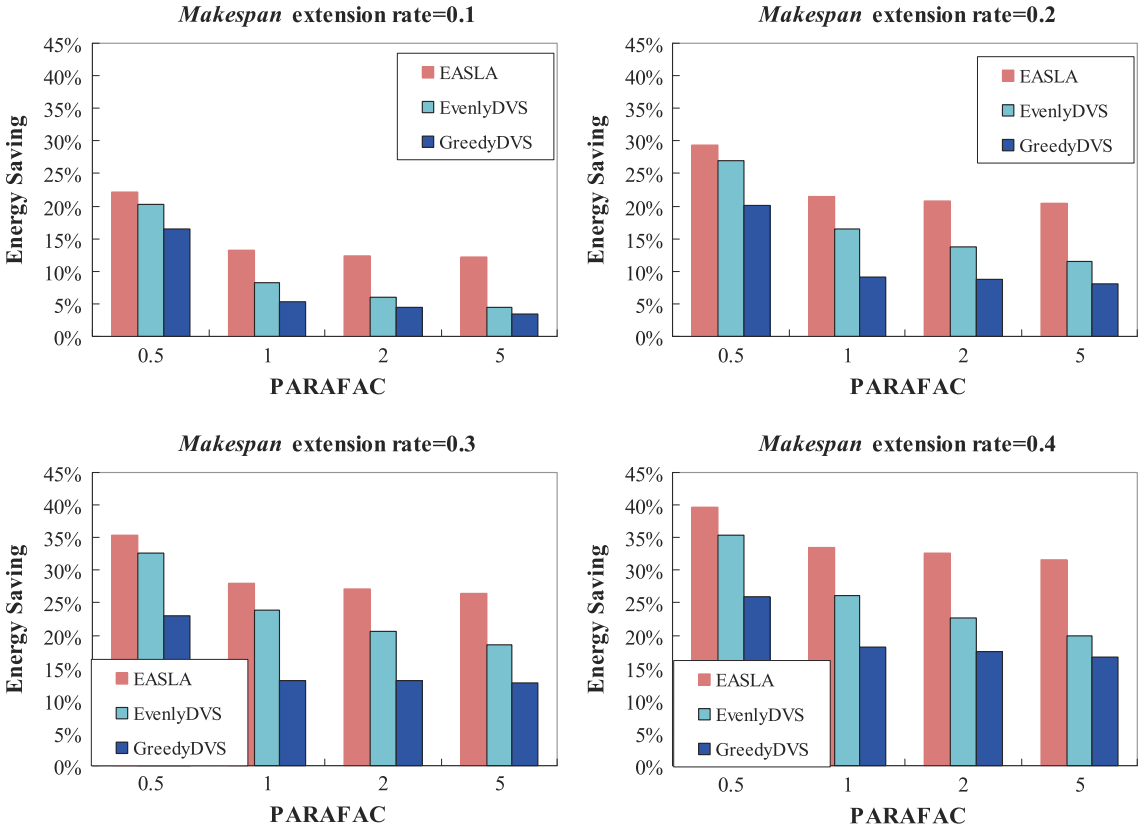


Fig. 8. Energy saving with respect to various parallelism degrees at different *makespan* extension rates (400 tasks, 10 PEs, *CCR* = 0.2).

algorithms by: (4.45%, 13.88%), (7.4%, 15.24%), (9.97%, 15.14%), and (11.73%, 14.94%), for parallelism degree of 0.5, 1, 2 and 5 respectively. It is also observed that the EASLA algorithm can get higher percentage improvement over the EvenlyDVS algorithm as the increase of the parallelism degrees. Since a high parallelism degree application has more independent tasks which is beneficial for our EASLA algorithm to take full advantage of slacks.

6.2.2. Comparison of energy saving for real-world applications

In this subsection, two real-world applications are tested. One is the application of a molecular dynamic code algorithm [7], which consists of 41 tasks. The other is the DSP application of sonar data processing [35], which consists of 119 tasks. The number of PEs is varied to 4 and 6 for the former application, and to 5 and 10 for the latter application.

Table 4 shows the percentage improvement of EASLA over EvenlyDVS and GreedyDVS for various values of *makespan* extension rates as well as various numbers of PEs for the molecular dynamic code algorithm and DSP based application. The results are consistent with the results for randomly generated applications. Our EASLA algorithm outperforms the other two algorithms for both applications. For instance, for 0.4 extension rate, the EASLA algorithm improves by 10.01%–12.01% compared with EvenlyDVS, and 12.36%–22.68% compared with GreedyDVS.

6.3. Performance results in heterogeneous environments

In this subsection, we use six sizes of random DAG tasks for our experiments, which are 30, 100, 200, 300, 500, and 1000. We choose three real PEs to simulate a heterogeneous environment and their voltage–frequency pairs are presented in Table 2. The number of PEs in the following experiments is set as {6, 30, 30, 30, 30, 60}, respectively. We compare the energy savings of the EASLA algorithm with the EES algorithm. As we do experiments in heterogeneous environments, we exchange the ETF algorithm to HEFT algorithm, which provides a good quality of schedules for DAGs in heterogeneous environments [30].

The results are shown in Fig. 9. Based on the observations from these figures, we can find that our EASLA algorithm can save more energy than the EES algorithm in any case. On the one

Please cite this article in press as: Y. Hu, et al., Slack allocation algorithm for energy minimization in cluster systems, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.08.022

Table 4
Percentage improvement of EASLA over EvenlyDVS and GreedyDVS for DAGs generated by real-world applications.

PEs			Makespan extension rate(η)					
			0.00	0.05	0.1	0.2	0.3	0.4
Molecular	4	EvenlyDVS	0.12	5.21	8.31	8.58	7.61	10.60
		GreedyDVS	0.66	6.90	11.37	17.63	20.44	20.77
	6	EvenlyDVS	0.41	5.50	8.55	9.17	8.11	10.02
		GreedyDVS	2.17	8.58	13.19	19.54	22.32	22.68
DSP	5	EvenlyDVS	0.05	4.14	6.88	7.61	7.57	10.01
		GreedyDVS	0.11	4.00	6.41	9.70	12.12	13.17
	10	EvenlyDVS	0.34	4.28	6.63	8.63	9.91	12.01
		GreedyDVS	0.49	3.93	5.68	8.58	11.14	12.36

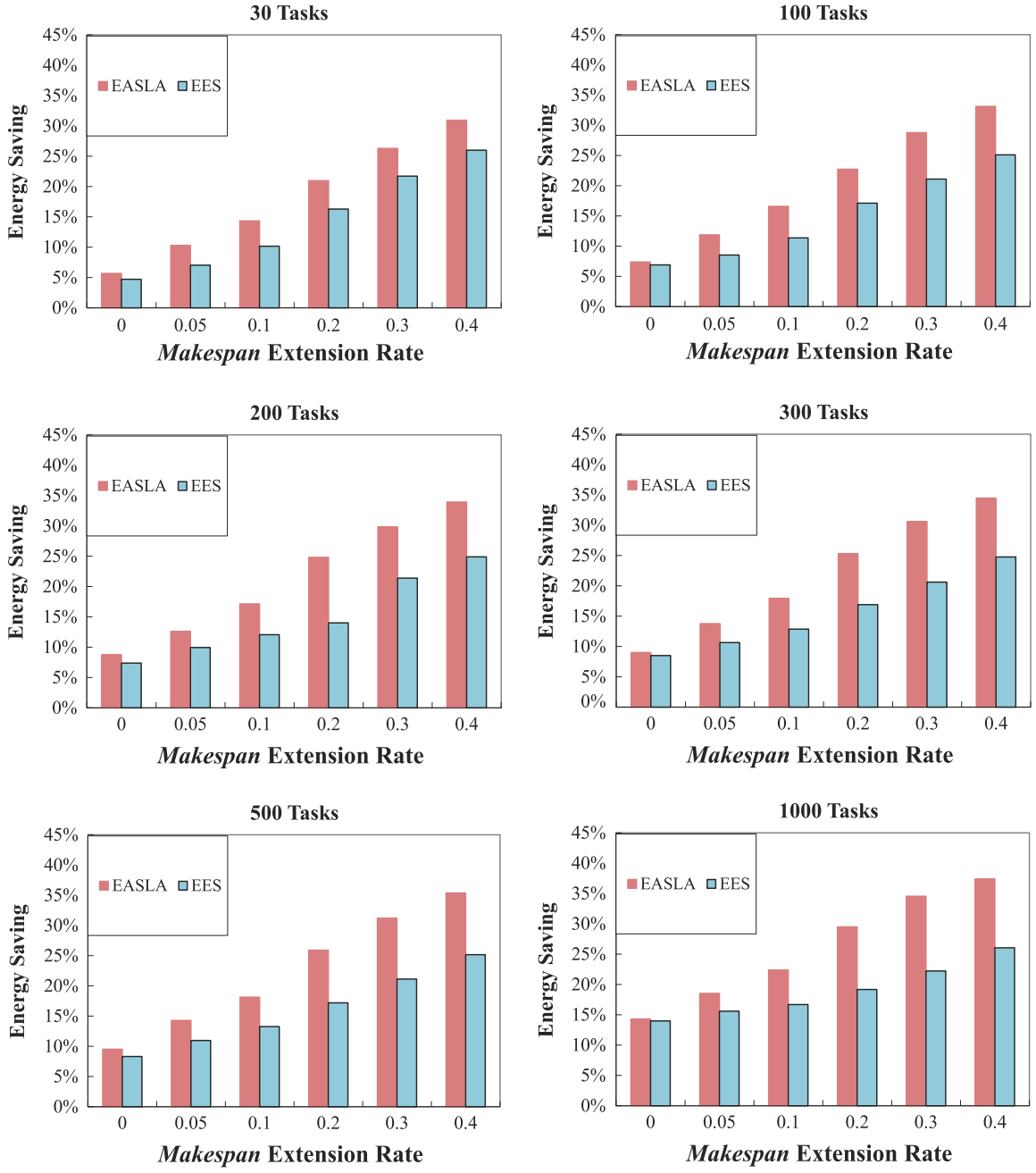


Fig. 9. Energy saving of EASLA and EES for various numbers of tasks and *makespan* extension rates in heterogeneous environments.

hand, the difference between the results of EASLA algorithm and EES algorithm tends to increase with the increase of *makespan* extension rate and our algorithm can save up to 12.33% energy consumption over EES when the *makespan* extension rate reaches 0.4 with 1000 tasks. On the other hand, at first, the saved energy also slightly increases with the increase of number of tasks (30,

Please cite this article in press as: Y. Hu, et al., Slack allocation algorithm for energy minimization in cluster systems, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.08.022

100, and 200). However, when the number of tasks is large enough, e.g., 200, 300, 500, the saved energy is almost kept unchanged. The reason behind lies in that when the number of tasks is large enough while the number PEs keeps the same, less slacks can be used to adjust frequency to save energy. Furthermore, when the number of PEs increases (see 1000 tasks with 60 PEs), the saved energy also tends to slightly increase.

7. Conclusion

A good energy aware scheduling strategy on DVFS-enabled cluster systems can reduce much energy consumption, which decreases the operational cost, reduces environmental pollution, and increases the system reliability. In this paper, we propose an energy-performance tradeoff scheduling algorithm EASLA, which tries to minimize energy consumption within certain extension rate of *makespan*. The proposed scheduling algorithm can be efficiently applied to applications that can be represented by a DAG. DAGs can represent a large number of applications. For instance, it can be used for DSP applications in sonar data processing, the molecular dynamic code algorithm, the Gaussian elimination algorithm, and decomposition of a large matrix. Randomly generated graphs and two real-world applications are adopted in our experiments to evaluate the performance of the proposed algorithm. Through the experimental results, EASLA can reduce energy consumption by up to 22.68% and 12.01% compared with GreedyDVS and EvenlyDVS algorithms in homogeneous environments, and 10.49% energy consumption compared with the EES algorithm in heterogeneous environments.

In the future work, we are planning to test the algorithm on variable degrees of heterogeneity among PEs and subtasks. Furthermore, we will consider individual components such as disks and memory to reduce energy consumption.

Acknowledgments

A preliminary version of this manuscript was presented on the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS 2014).

The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), the National Natural Science Foundation of China (Grant Nos. 61370095, 61472124), the National High-tech R&D Program of China (Grant No. 2014AA01A302), and the International Science & Technology Cooperation Program of China (Grant No. 2015DFA11240, 2014DFBS0010).

References

[1] Y.C. Lee, A.Y. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, IEEE Trans. Parallel Distrib. Syst. 22 (8) (2011) 1374–1381.
[2] X. Zhu, C. He, K. Li, X. Qin, Adaptive energy-efficient scheduling for real-time tasks on DVS-enabled heterogeneous clusters, J. Parallel Distrib. Comput. 72 (6) (2012) 751–763.
[3] Y. Ma, B. Gong, L. Zou, Energy-optimization scheduling of task dependent graph on DVS-enabled cluster system, in: ChinaGrid Conference (ChinaGrid), 2010 Fifth Annual, IEEE, 2010, pp. 183–190.
[4] G. von Laszewski, L. Wang, Greenit service level agreements, in: Grids and Service-Oriented Architectures for Service Level Agreements, Springer, 2010, pp. 77–88.
[5] D. Gmach, Y. Chen, A. Shah, J. Rolia, C. Bash, T. Christian, R. Sharma, Profiling sustainability of data centers, in: 2010 IEEE International Symposium on Sustainable Systems and Technology, (ISSST), IEEE, 2010, pp. 1–6.
[6] Y. Zhang, X.S. Hu, D.Z. Chen, Task scheduling and voltage selection for energy minimization, in: Proceedings of the 39th Annual Design Automation Conference, ACM, 2002, pp. 183–188.
[7] J. Mei, K. Li, K. Li, Energy-aware task scheduling in heterogeneous computing environments, Cluster Comput. (2013) 1–14.
[8] S. Baskiyar, R. Abdel-Kader, Energy aware dag scheduling on heterogeneous systems, Cluster Comput. 13 (4) (2010) 373–383.
[9] D. Zhu, R. Melhem, B. Childers, Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems, IEEE Trans. Parallel Distrib. Syst. 14 (7) (2003) 686–700.
[10] J. Kang, S. Ranka, Slack allocation algorithm for parallel machines, J. Parallel Distrib. Comput. 70 (1) (2010) 23–34.
[11] G. Aupy, A. Benoit, Y. Robert, Energy-aware scheduling under reliability and makespan constraints, in: 2012 19th International Conference on High Performance Computing, (HiPC), IEEE, 2012, pp. 1–10.
[12] S. Baskiyar, K.K. Palli, Low power scheduling of dags to minimize finish times, in: High Performance Computing-HiPC 2006, Springer, 2006, pp. 353–362.
[13] C. Liu, K. Li, C. Xu, K. Li, Strategy configurations of multiple users competition for cloud service reservation, IEEE Trans. Parallel Distrib. Syst. 27 (2) (2016) 508–520.
[14] H. Kimura, M. Sato, Y. Hotta, T. Boku, D. Takahashi, Emprical study on reducing energy of parallel programs using slack reclamation by dvfs in a power-scalable high performance cluster, in: 2006 IEEE International Conference on Cluster Computing, IEEE, 2006, pp. 1–10.
[15] N.B. Rizvandi, J. Taheri, A.Y. Zomaya, Y.C. Lee, Linear combinations of dvfs-enabled processor frequencies to modify the energy-aware scheduling algorithms, in: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, (CCGrid), IEEE, 2010, pp. 388–397.
[16] P. Chowdhury, C. Chakrabarti, Static task-scheduling algorithms for battery-powered DVS systems, IEEE Trans. Very Large Scale Integrat. (VLSI) Syst. 13 (2) (2005) 226–237.
[17] L. Wang, S.U. Khan, D. Chen, J. Kołodziej, R. Ranjan, C.-z. Xu, A. Zomaya, Energy-aware parallel task scheduling in a cluster, Future Gener. Comput. Syst. 29 (7) (2013) 1661–1670.
[18] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, X. Huang, Enhanced energy-efficient scheduling for parallel applications in cloud, in: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, (ccgrid 2012), IEEE Computer Society, 2012, pp. 781–786.
[19] M. Mezmaz, N. Melab, Y. Kessaci, Y.C. Lee, E.-G. Talbi, A.Y. Zomaya, D. Tuyttens, A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems, J. Parallel Distrib. Comput. 71 (11) (2011) 1497–1508.
[20] C.-M. Wu, R.-S. Chang, H.-Y. Chan, A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters, Future Gener. Comput. Syst. 37 (2014) 141–147.
[21] K.H. Kim, R. Buyya, J. Kim, Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters., in: CCGRID, Vol. 7, 2007, pp. 541–548.
[22] Y. Gao, H. Guan, Z. Qi, T. Song, F. Huan, L. Liu, Service level agreement based energy-efficient resource management in cloud data centers, Comput. Electr. Eng. 40 (2014) 1621–1633.
[23] M. Marzolla, R. Mirandola, Dynamic power management for qos-aware applications, Sustain. Comput.: Inf. Syst. 3 (4) (2013) 231–248.
[24] M.E. Haque, K. Le, Í. Goiri, R. Bianchini, T.D. Nguyen, Providing green slas in high performance computing clouds, in: Green Computing Conference (IGCC), 2013 International, IEEE, 2013, pp. 1–11.
[25] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768.
[26] J. Cao, K. Hwang, K. Li, A.Y. Zomaya, Optimal multiserver configuration for profit maximization in cloud computing, IEEE Trans. Parallel Distrib. Syst. 24 (6) (2013) 1087–1096.
[27] S. Tang, J. Yuan, C. Wang, X.-Y. Li, A framework for amazon ec2 bidding strategy under sla constraints, IEEE Trans. Parallel Distrib. Syst. 25 (1) (2014) 2–11.
[28] R. Ge, X. Feng, K.W. Cameron, Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters, in: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, IEEE Computer Society, 2005, p. 34.
[29] H. El-Rewini, H.H. Ali, T. Lewis, Task scheduling in multiprocessing systems, Computer 28 (12) (1995) 27–37.
[30] H. Topcuoglu, S. Hariri, M.-y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.
[31] J. Luo, N.K. Jha, Power-profile driven variable voltage scaling for heterogeneous distributed real-time embedded systems, in: 16th International Conference on VLSI Design, 2003. Proceedings, IEEE, 2003, pp. 369–375.
[32] S. Sakai, M. Togasaki, K. Yamazaki, A note on greedy algorithms for the maximum weighted independent set problem, Discrete Appl. Math. 126 (2) (2003) 313–322.
[33] H. Yu, Y. Ha, B. Veeravalli, Quality-driven dynamic scheduling for real-time adaptive applications on multiprocessor systems, IEEE Trans. Comput. 62 (10) (2013) 2026–2040.
[34] Y. Xu, K. Li, L. He, T.K. Truonn, A dag scheduling scheme on heterogeneous computing systems using double molecular structure-based chemical reaction optimization, J. Parallel Distrib. Comput. 73 (9) (2013) 1306–1322.
[35] C.M. Woodside, G.G. Monforton, Fast allocation of processes in distributed and parallel systems, IEEE Trans. Parallel Distrib. Syst. 4 (2) (1993) 164–174.

Please cite this article in press as: Y. Hu, et al., Slack allocation algorithm for energy minimization in cluster systems, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.08.022

Yikun Hu is currently working towards the Ph.D. degree at Hunan University, China. His research interests are mainly in parallel and distributed processing, Cluster, Grid and Cloud computing.

Chubo Liu is currently working toward the Ph.D. degree at Hunan University, China. His research interests are mainly in modeling and scheduling for distributed computing systems, Approximation and random algorithms, Grid and Cloud computing. He is a programming lover and interested in design of algorithms. Game theory is also included in his current research interests.

Kenli Li received the Ph.D. in computer science from Huazhong University of Science and Technology, China, in 2003, and the M.Sc. in mathematics from Central South University, China, in 2000. He was a visiting scholar at University of Illinois at Champaign and Urbana from 2004 to 2005. Now he is a professor of Computer science and Technology at Hunan University, associate director of National Supercomputing Center in Changsha, a senior member of CCF. His major research includes parallel computing, high-performance computing, Grid and Cloud computing. He has published more than 150 research papers in international conferences and journals such as IEEE-TC, IEEE-TPDS, IEEE-TSP, JPDC, ICPP, CCGrid. He is an outstanding member of CCF. He is a senior member of the IEEE and serves on the editorial board of IEEE Transactions on Computers.

Xuedi Chen is currently working toward the M.S. degree at Hunan University of China. Her research interests include scheduling for distributed computing systems and parallel algorithms.

Keqin Li is a SUNY Distinguished Professor in computer science. Now he is also a professor of Computer science and Technology at Hunan University. His research interests are mainly in design and analysis of algorithms, parallel and distributed computing, and computer networking. He has contributed extensively to processor allocation and resource management; design and analysis of sequential/parallel, deterministic/probabilistic, and approximation algorithms; parallel and distributed computing systems performance analysis, prediction, and evaluation; job scheduling, task dispatching, and load balancing in heterogeneous distributed systems; dynamic tree embedding and randomized load distribution in static networks; parallel computing using optical interconnections; dynamic location management in wireless communication networks; routing and wavelength assignment in WDM optical networks; energy-efficient power management and performance optimization. His current research interests include lifetime maximization in sensor networks, file sharing in peer-to-peer systems, and cloud computing. He has published over 235 journal articles, book chapters, and research papers in refereed international conference proceedings. He has received several Best Paper Awards for his highest quality work. He is currently on the editorial board of *IEEE Transactions on Parallel and Distributed Systems, Journal of Parallel and Distributed Computing, International Journal of Parallel, Emergent and Distributed Systems, International Journal of High Performance Computing and Networking, and Optimization Letters.*

Please cite this article in press as: Y. Hu, et al., Slack allocation algorithm for energy minimization in cluster systems, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.08.022