Energy aware list-based scheduling for parallel applications in cloud

Yongxing Liu, Kenli Li*, Zhuo Tang and Keqin Li

College of Computer Science and Electronic Engineering,
Hunan University,
Changsha, 410082, China
Email: yongxing510@126.com
Email: lkl@hnu.edu.cn
Email: ztang@hnu.edu.cn
Email: likq@hnu.edu.cn
*Corresponding author

Abstract: As the growth of energy consumption has been explosive in current data centres and cloud systems, it has drawn greater attention in academia, industry and government. Task scheduling as a core in systems, it has become an important method to reduce energy dissipation. This paper proposes an energy aware list-based scheduling algorithm called EALS for parallel applications in the context of service level agreement (SLA) on cloud data centres. First, the EALS algorithm comprehensively considers the high power processors to minimise the number of high power processors used. Then, the algorithm try to migrate some tasks from a high power processor to a low power processor for energy saving. Finally, the EALS algorithm takes a more efficient way to assign the time slots among tasks based on the dynamic voltage scaling (DVS) technique. To demonstrate the effectiveness of the EALS algorithm, randomly generated graphs and several real-world applications are tested in our experiments. The experimental results show that the EALS algorithm can save up to 43.96% energy consumption for various parallel applications as well as balance the scheduling performance.

Keywords: cloud data centre; directed acyclic graph; dynamic voltage scaling; DVS; energy aware scheduling; service level agreement; SLA.

Reference to this paper should be made as follows: Liu, Y., Li, K., Tang, Z. and Li, K. (xxxx) 'Energy aware list-based scheduling for parallel applications in cloud', *Int. J. Embedded Systems*, Vol. X, No. Y, pp.xxx–xxx.

Biographical notes: Yongxing Liu is currently working towards his MS at Hunan University of China. His research interests include modelling and scheduling for embedded systems, distributed computing systems, and cloud computing.

Kenli Li received his PhD in Computer Science from Huazhong University of Science and Technology, Wuhan, China in 2003. He has been a Visiting Scholar at University of Illinois at Champaign and Urbana from 2004 to 2005. His major research contains parallel computing, grid and cloud computing, and DNA computer.

Zhuo Tang is currently an Associate Professor of Computer Science and Technology at Hunan University. He received his PhD in Computer Science from Huazhong University of Science and Technology, Wuhan, China in 2008. His current research interests include cloud computing, distributed systems and security of distributed systems.

Keqin Li is a SUNY Distinguished Professor of Computer Science in the State University of New York at New Paltz. His research interests are mainly in design and analysis of algorithms, parallel and distributed computing, and computer networking. He is currently on the editorial board of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *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*.

This paper is a revised and expanded version of a paper entitled [title] presented at [name, location and date of conference].

Comment [t1]: Author: Please complete where highlighted.

1 Introduction

The demand of social life prompts the emergency and development of information science, while the development of information science on human society life, production and technological progress plays a huge role. However, too much of a good thing can also be bad, the cost and operational expenses of cloud data centres have soared with the lasting increase in computing capacity. Energy consumption has increasingly become a critical concern for cloud data centres owing to their high operation cost, environmental impact, and low reliability. So it is significant to study the strategy of reducing energy consumption in data centres.

Energy awareness for parallel applications has become a growing concern for a decade. There are two popular mechanisms for energy saving in data centres: the dynamic voltage scaling (DVS) (Mittal, 2014; Mehta and Amrutur, 2012) and the dynamic power management (DPM) (Benini et al., 2000; Liang et al., 2013). The former achieves massive energy savings by dynamically adjusting processor supply voltages based on the computing load; and the latter optimises energy consumption by powering off the idle computing nodes at runtime. In order to use the DPM more efficiently, a scheduling scheme must consolidate tasks on a minimum set of computing nodes to power off some nodes. For the scheduling of precedence-constrainted applications, the DPM is not quite suitable for this case, because the idle slot between the execution of two tasks is usually short. Instead, the DVS is particularly suited for it. The problem of scheduling an application, usually expressed as directed acyclic graph (DAG) scheduling, is NP-hard in the general case (Garey and Johnson, 1979; Shao et al., 2004). Therefore, one of the challenges in computing systems is to develop scheduling that allocate the appropriate CPU time to the tasks of an application, especially with energy awareness in cloud data centres. There is a rich body of studies and investigations pertaining to energy savings use of the DPM or the DVS in computing systems. However, most of these approaches just take one of the mentioned technologies to reduce energy consumption. Consequently, these approaches cannot fully exploit the latent capacity of the two technologies, so these approaches consume more resources in computing systems.

In this paper, we address the energy-aware scheduling of precedence-constrained parallel applications on a heterogeneous computing (HC) system and propose a novel scheduling algorithm that combines the DVS and the DPM, namely energy aware list-based scheduling (EALS) algorithm. Our objective is to minimise the total energy consumption of the tasks while it still meets the certain performance goals-based service level agreement (SLA) (Wang et al., 2013; Wu et al., 2014; Quan, 2007). The SLA is a contract between users and their service providers, guaranteeing the quantifiable quality of service at defined levels. Our algorithm mainly contains three stages:

1 initial task mapping phase stage: ordering all tasks according to calculated priorities and assigning the tasks to corresponding processors

2 DPM optimising and task migrating optimisation phase: powering off the processors guided by the ideology of DPM and migrating some of high energy-consuming tasks to a appropriate processor

3 task slacking phase: scaling the frequency of tasks with a heuristic method.

The main contributions of this paper are summarised as follows:

- We develop a novel energy-aware scheduling algorithm. Within a given deadline, the proposed algorithm can distribute the tasks to a set of appropriate processors, reassign part of tasks, and allocate the time slots among tasks in an effective manner to reduce the total energy consumption as well as meet the performance requirements.
- We demonstrate the effectiveness of EALS algorithm through extensive experiments. Experimental results show that the proposed algorithm can achieve outstanding energy-saving effects in a wide range of applications.
- We analyse the factors which are affecting the performance of our algorithm.

The rest of this paper is organised as follows. Section 2 introduces background and related works. We describe the related model in Section 3. Section 4 presents the detail of our scheduling algorithm EALS. In Section 5, we compare our experimental results with two related algorithms. Finally, we conclude this paper and give an overview of future work in Section 6.

2 Related work

In this section, we review background and related work of traditional task scheduling and energy-aware scheduling in computing systems.

2.1 Traditional task scheduling

The problem of optimisation scheduling of HC systems has been researched extensively due to its high academic and practical significance. There are several typical classifications of DAG scheduling: list-based scheduling algorithms, cluster heuristics algorithms and duplication-based algorithms, etc. The heterogeneous earliest finish time (HEFT) is a famous list-based algorithm with high performance and low complexity for HC systems (Topcuoglu et al., 2002). Many other list-based algorithms, such as modified critical path (MCP) (Wu and Gajski, 1990), dynamic critical path (DCP) (Kwok and Ahmad, 1996) and critical path on a processor (CPOP) (Topcuoglu et al., 2002) are classical. Cluster heuristics divide tasks of

an application into several subsets and then performs an order over each subsets individually. Some examples in this classification include dominant sequence cluster (DSC) (Yang and Gerasoulis, 1994), greedy task clustering and scheduling (GTCS) (Piyatamrong et al., 2000), clustering heuristic scheduling algorithm (CHSA) (Ilyas and Khan, 2001), and so on. In duplication-based algorithms, heterogeneous critical parents with fast duplicator (HCPFD) (Hagras and Janeček, 2005), selective duplication (SD) (Bansal et al., 2003), heterogeneous limited duplication (HLD) (Bansal et al., 2005) and heterogeneous earliest finish with duplication (HEFD) (Tang et al., 2010) are able to perform well in a system.

2.2 Energy-aware scheduling

In consideration of the serious energy-using situation and the lasting rapid growth in energy consumption of computing systems, energy-saving has become increasingly important. Therefore, lots of researchers have done much research work for reducing energy consumption for computing systems (Mei and Li, 2012; Zhong and Xu, 2007; Xiao and Han, 2014; Huang et al., 2012; Niu and Quan, 2013; Dargie, 2012; Wang et al., 2013). Mei and Li (2012) proposed an energy-aware scheduling by minimising duplication (EAMD) algorithm. Dargie (2012) investigated aspects of power dissipation in a node; analysed the strengths and weaknesses of DVS and provided a comprehensive assessment of the DPM. Zhong and Xu (2007) discussed scheduling tasks with DVS on single-processor systems and obtained good results. For dependent real-time tasks, the blocking-time stealing (BTS) algorithm can able to achieve energy saving and it still satisfies the time constraints of tasks (Wu and Wu, 2014). Mei et al. (2014) proposed a resource-aware scheduling algorithm with duplications (RADS) to search and delete redundant task duplications dynamically during the scheduling process. Huang et al. (2012) developed an enchanted energy-efficient scheduling (EES) algorithm with SLA to save energy consumption. To reduce both the dynamic and leakage energy consumption, two energy efficient scheduling approaches are proposed (Niu and Quan, 2013). Wang et al. (2013) proposed a power aware task clustering (PATC) algorithm, a power aware list-based scheduling (PALS) algorithm and an energy-performance tradeoff scheduling (ETS) algorithm, the effectiveness of their algorithms are justified by a simulation study.

3 Models

In this section, we introduce a computing system model and an application model.

3.1 Computing system model

The system comprises a set P of m heterogeneous processors, which are fully interconnected by a communication links. And our target system is denoted by $P = \{p_i | 0 \le i \le m - 1\}$, where p_i is DVS-enabled and it runs in varying clock speeds. Three types of processors are considered in our simulation experiments and each processor has its own performance state (PState), which is shown in Table 1 (Terzopoulos and Karatza, 2013). Factors like the processor architecture, the task processing requirement and their compatibility decide the capacity of a processor in processing a task.

Table 1 Performance and power consumption

Frequency (GHz)	*P-state-watts*		
	AMD Opteron	*Intel Pentium M*	*VIA C7-M*
2.6	P0–95	-	-
2.4	P1–90	-	-
2.2	P2–76	-	-
2.0	P3–65	-	P0–20
1.8	P4–55	-	P1–18
1.6	-	P0–25	P2–15
1.4	-	P1–17	P3–13
1.2	-	P2–13	-
1.0	P5–32	P3–10	P4–10
0.8	-	P4–8	P5–7
0.6	-	P5–6	P6–6
0.4	-	-	P7–5
Idle	15	5	0.1

Table 1 shows that the energy consumption model is different from a continuous energy model, which is that each processor of our system run only on a particular frequency points of the set and we cannot change the frequency of a processor continuously. The energy consumption of processor p_i is mainly comprise of active energy consumption $E^i_{p,active}$ and idle energy consumption $E^i_{p,idle}$. So the total energy consumption of a system can be defined as:

$$E_{total} = \sum_{i=0}^{m-1} \left(E^i_{p,active} + E^i_{p,idle} \right). \tag{1}$$

3.2 Application model

In general, a parallel application can be represented by a DAG and we can use $G = G(V,E, \Omega, \Psi)$ to formulate it, where $V = \{v_i | 0 \le i \le n - 1\}$ represents the tasks of an application, $E = \{e_{i,j} | 0 \le i, j \le n - 1\}$ represents the dependencies among tasks, $\Omega = \{\omega_{i,k} | 0 \le i \le n - 1, 0 \le k \le m - 1\}$ denotes the computation cost of task v_i on processor p_k, and $\Psi = \{\psi(e_{i,j}) | 0 \le i, j \le n - 1\}$ denotes the communication cost between task v_i and v_j. A task with no parent is called an entry task v_{entry}, and v_{exit} represents an exit task with no children. A parent which completes the communication at latest time is called the most influential immediate parent (MIIP) of the task expressed as $v_{miip(v_i)}$.

Obviously, $eft(v_i, p_k)$, the earliest finish time of task v_i on processor p_k, is mainly determined by the $v_{miip(v_i)}$. The earliest start time, $est(v_i, p_k)$, and the $eft(v_i, p_k)$ of a task v_i on a processor p_k can be defined as:

$$est(v_i, p_k) = \begin{cases} 0 & \text{, if } v_i = v_{entry}; \\ max\left(\begin{array}{l} \min\left(idle^n_{k.available}.start\right), \\ \max\left(est(v_\varsigma, p_l) + \omega_{i,k} + \psi(e_{\varsigma,i})\right) \\ v_\varsigma \in pred(v_i) \end{array}\right) & \text{, otherwise;} \end{cases} \quad (2)$$

$$eft(v_i, p_k) = est(v_i, p_k) + \omega_{i,k}, \quad (3)$$

where $idle^n_{k.available}.start$ is the start time of the n^{th} idle slack meeted $idle^n_{k.available}.end - idle^n_{k.available}.start \geq \omega_{i,k}$, $pred(v_i)$ is the set of immediate parent of vi, and if $p_k = p_l$, then $\psi(e_{\varsigma,i}) = 0$.

A list-based scheduling algorithm usually generates a task priority in first and orders all tasks according to calculated priorities prior to assignment. In this paper, we adopt the bottom level (*b level*) method to calculate the task priority. The *b level* of a task v_i is the length of a longest path from v_i to an exit task (Kwok and Ahmad, 1999). Let $rank_b(v_i)$ represent the *b level* value of task v_i, then $rank_b(v_i)$ can be recursively calculated by

$$rank_b(v_i) = \begin{cases} \overline{\omega}_{exit} & \text{,if } v_i = v_{exit}; \\ \overline{\omega_i} + \max\limits_{v_\sigma \in succ(v_i)} \left(\psi(e_{i,\sigma}) + rank(v_\sigma)\right) & \text{, otherwise;} \end{cases} \quad (4)$$

where $succ(v_i)$ is the set of immediate children of v_i, and $\overline{\omega_i}$ is the average computation cost of task v_i. The $\overline{\omega_i}$ value of task v_i can be calculated by

$$\overline{\omega_i} = \frac{1}{m}\sum_{k=0}^{m-1} \omega_{i,k}, \quad (5)$$

where m is the number of processors.

There will probably be a lot of idle slots among tasks on processors due to the precedence constraints. We can optimise the total energy consumption by taking full advantage of these slots. In the process of energy optimisation, we usually need to calculate the latest finish time of tasks. The latest finish time of task v_i on a processor p_k can be defined as

$$lft(v_i, p_k) = \begin{cases} makespan & \text{, if } v_i = v_{exit}; \\ \min\left(\begin{array}{l} lft(v_\tau, p_k) - \omega_{\tau,k}, \\ \min\left(lft(v_\sigma, p_l) - \omega_{\sigma,p_l} - \psi(e_{i,\sigma})\right) \\ v_\sigma \in succ(v_i) \end{array}\right) & \text{, otherwise;} \end{cases} \quad (6)$$

where *makespan* is the schedule length of parallel applications, v_τ is the task assigned next to v_i on the same processor p_k, v_σ is the task assigned on the processor p_l, and if $p_k = p_l$, then $\psi(e_{i,\sigma}) = 0$. Then, the latest start time of task v_i on a processor p_k can be calculated by

$$lst(v_i, p_k) = lft(v_i, p_k) - \omega_{i,k}. \quad (7)$$

4 Proposed algorithm

In this section, we present the details of our algorithm EALS. As the overall scheduling processes in EALS, Algorithm 1 aims at minimising the total energy consumption as much as possible with the determined SLA constrain. The EALS algorithm has three major phases:

1 initial task mapping phase

2 DPM optimising and task migrating optimisation phase

3 task slacking phase.

Algorithm 1 Energy aware list-based scheduling

Require: μ, P, G.

Ensure: $\Im$. ▷ A Schedule,

1 **call** Algorithm 2 to generate the initial schedule $\Im$,

2 **call** Algorithm 3 to optimise the schedule and output the intermediate result $\Im'$;

3 **call** Algorithm 5 to further optimise the schedule and update the result $\Im$.

4.1 Initial task mapping phase

This subsection discusses the initial allocation of the tasks. The most important work in initialisation is to calculate the task priority, order the tasks, and schedule one by one. Here we employ the Algorithm 2 to map the tasks of an application to the processor set P.

The output of Algorithm 2 can be considered as a initial schedule, which is taken as an input of the next optimising process.

Algorithm 2 Task mapping

Require: P, G.

Ensure: $\Im$, £. ▷ $\Im$: A Schedule, £: makespan,

```
for each task v_i in G do
    calculate ω̄_i by equation (5),
end for
for each task v_i in G do
    calculate rank_b(v_i) by equation (4),
end for
rank the tasks into a sequence by a non-decreasing order based on b level and let RANK represent the sequence,
while RANK is not null do
    task v_i = RANK.pop(),
    for each processor p_k in P do
        calculate est(v_i, p_k) by equation (2),
        calculate eft(v_i, p_k) by equation (3),
    end for
    assign task v_i to the processor p_k that minimises finish time of task v_i,
end while
```

4.2 DPM optimising and task migrating optimisation phase

Considering the lasting increase of energy consumption in cloud data centres, reducing the total energy consumption is imminent in systems. So green computing has been paid attention increasingly in recent years. A SLA, contracted between a service provider and a user, is used to guarantee quantifiable performance at defined levels. In the context of scheduling, a SLA means an acceptable performance loss to users, that is, a service provider can reduce the total energy consumption by extending the *makespan* in a system. Scheduling tasks without increasing the *makespan* can be referred as the 'best-effort scheduling issue', let $makespan_{best}$ represent the *makespan* of this case. In this paper, finding a feasible schedule which tries to minimise the total energy consumption subjecting to $makespan \leq (1 + \mu) \times makespan_{best}$, where μ is the makespan extension factor determined by the SLA and meeted $\mu \geq 0$. The detailed optimisation process is presented by Algorithm 3.

In order to get better energy saving effects, the effective computation score of each processor is calculated by equation (8).

$$ecs(p_i) = \frac{type(p_i)^{watt}_{level-0}}{type(p_i)^{frequency}_{level-0}}, \quad (8)$$

where $type(p_i)$ is one of the three processor types in our system model, $type(p_i)^{watt}_{level-0}$ represents the power consumption of processor p_i at level-0, and $type(p_i)^{frequency}_{level-0}$ is the corresponding frequency at level-0. Having calculated the $ecs(p_i)$ of each processor p_i, we sort $ecs(p_i)$s into a sequence by a non-increasing order of effective computation score. Note that in our system model, the lower the $ecs(p_i)$ is, the more effective the processor is.

Algorithm 3 Optimising a schedule with DPM

Require: $\Im$, £, μ, P, G.

Ensure: $\Im'$. ▷ $\Im'$: Intermediate Schedule,

```
for each p_i in P do
    calculate ecs(p_i) by equation (8),
end for
sort ecs(p_i)s with non-increasing order, let ecsSeq denote this sequence,
let currProType = type(ecsSeq.peekFirst),
    ▷current processor type,
while makespan ≤ £ × (μ + 1) do
    pick a processor p_k belonging to the currProType type from the ecsSeq,
    power off the processor p_k,
    call Algorithm 2 to reschedule the tasks,
    if makespan ≤ £ × (μ + 1) then
        update the ℑ with current assignments,
    else
        power on the processor p_k,
        currPro ype + +,
    end if
    if all processors of currProType type are powered off then
        currProType + +,
    end if
    if currProType does not exist then
        break,
    end if
end while
call Algorithm 4 to generate the output, ℑ';
```

In Algorithm 3, the processor p_i with a high $ecs(p_i)$ value can be considered one of the first attempts to power off. If the processor p_i is powered off, and it still meets the SLA constraint, then we can keep trying to power off other processors which belong to the current processor type, otherwise we have to undo this operation and try to power off a processor with a lower *ecs* value until all processor types are handled successfully. One advantage of Algorithm 3 is that the approach can minimise the number of high power processors used. Another advantage of Algorithm 3 is that the approach can achieve higher energy efficiency by Algorithm 4.

Having finished the process of powering off, some tasks are migrated from an inefficient processor to an efficient processor by the Algorithm 4. In the process of migration, if a task is placed on the other processor can bring an energy saving effect, then the task should be migrated.

Algorithm 4 Migrating tasks

Require: $\Im$, £, μ, P, G.

Ensure: $\Im'$.

```
mark all tasks as unexamed,
update the latest finish time of task v_exit to £ × (μ + 1),
rank the finish time of tasks into a sequence by a
non-decreasing order and let FT represent the sequence,
while FT is not null do
    task v_i = FT.pop(),
    let p_k represent the assigned processor to task v_i,
    calculate lft(v_i, p_k) by equation (6),
    calculate lst(v_i, p_k) by equation (7),
end while
while ℑ is not null do
    popup the unexamed task v_i with the earliest start time
    from ℑ,
    let p_k denote the current assigned processor for v_i,
    for each p_n in P do
        if the processor p_n is powered off then
            continue,
        else if assigning p_n to v_i meets equation (9) then
            let p_k = p_n,
        end if
    end for
    assign task v_i to the processor p_k,
    mark task v_i as examed,
    mark v_miip(v_i),
end while
```

The migratable task v_i assigned on the processor p_k is the one which satisfies equation (9):

$$\begin{cases} \dfrac{\omega_{i,n}\times type(p_n)^{watt}_{level-0}}{\omega_{i,k}\times type(p_k)^{watt}_{level-0}} + \dfrac{(\omega_{i,n}-\omega_{i,k})\times type(p_n)^{watt}_{idle}}{\omega_{i,k}\times type(p_k)^{watt}_{idle}} \le 1; \\ eft(v_i,\ p_n)\ + \psi(e_{i,\sigma}) \le lst(v\_,\ p_l); \\ \qquad\qquad v_\sigma \in succ(v_i) \\ eft(v_i,\ p_n) \le lst(v_\tau,\ p_n); \end{cases} \tag{9}$$

where p_n is the new processor mapped for v_i, $type(p_n)^{watt}_{idle}$ is the idle power value of p_n, if $p_l = p_n$, then $\psi(e_{i,\sigma}) = 0$, and v_τ is the earliest unexamed task assigned on the processor p_n.

In Algorithm 4, each task v_i is examined one by one, if there is a processor p_n which matches with the task v_i more efficient and meets the constraints, then the Algorithm 4 will update the processor of the match with p_n, otherwise it will keep the original match unchanged. Moreover, the MIIP of each task v_i is recorded to provider convenience for post-process. Having done that, all migratable tasks are processed and the current process of the optimisation is finished. As a supplement of the EALS algorithm, the Algorithm 4 will be able to help the EALS get a better performance in communication bounded or low parallelism applications.

4.3 Task slacking phase

Having finished the first two phases of EALS, we can get an intermediate schedule of the DAG. Algorithm 5 aims to minimise the total energy consumption by slacking the tasks of an application. In order to calculate the slack time between two tasks, we first need to obtain the latest finish time of the tasks by equation (6). In Algorithm 5, one of the most important operations is to determine the frequency level of each task. Considering the earliest start time of each task v_i is always overlapped with the latest finish time of task v_j which is the latest task assigned ahead of v_i on the same processor p_k, and the start time of task v_i is subject to the task $v_{miip(v_i)}$, we have to balance the constraints to obtain a good energy saving effect. So we update the slack time of task being processed by equation (10).

$$\begin{aligned} &uslack(v_i) \\ &= lft(v_i,\ p_k) - \omega_{i,k} \\ &\quad - max\left\{ max\ \left\langle est(v_i,\ p_k), est(v_i,\ p_k) \right.\right. \\ &\quad + \left.\left. \frac{lft(v_j,\ p_k) - est(v_i,\ p_k)\times\omega_{i,k}}{\omega_{i,k}+\omega_{j,k}} \right\rangle, \right. \\ &\quad \left. \begin{cases} est(v_i,\ p_k) & , \text{ if } ecs(p_l) < ecs(p_k); \\ lft\left(v_{miip(v_i)},\ p_l\right) + \psi(e_{miip(vi),i}), & \text{otherwise;} \end{cases} \right\}, \end{aligned} \tag{10}$$

where v_j is the latest task assigned ahead of v_i on p_k, p_l is the processor assigned to $v_{miip(v_i)}$. Once the $uslack(v_i) > 0$ is determined, the ideal operating frequency of v_i on the processor p_k can be calculated by

$$f_{ideal}(v_i) = \frac{\omega_{i,k}\times type(p_k)^{frequency}_{level-0}}{\omega_{i,k}+uslack(v_i)}, \tag{11}$$

Then the running frequency of v_i can be determined by

$$f_{run}(v_i) = min\left(type(p_k)^{frequency}_{level-n}\right) \ge f_{ideal}(v_i), \quad n \in P - State \tag{12}$$

where $P - State$ is the set of frequency levels. Consequently, the execution time of task v_i is updated to

$$T_{exe}(v_i) = \omega_{i,k} \times \frac{type(p_k)^{frequency}_{level-0}}{f_{run}(v_i)}. \tag{13}$$

So, the start and finish time of task v_i on the processor p_k can be updated to $[lft(v_i, p_k) - T_{exe}(v_i), lft(v_i, p_k)]$.

From Algorithm 5 we can see that the unprocessed task v_i with largest $lft(v_i, p_k)$ is processed first in a loop, where p_k is the processor assigned to v_i. Then the slack time of task v_i is determined by equation (10). If the task v_i involves overlapping with the nearest task v_j assigned ahead of v_i, then the overlapped area is proportioned by their computation cost. In the meanwhile, if the $v_{miip(v_i)}$ is

assigned on a processor with higher effective computation score, the equation (10) will allocate more space to the $v_{miip(v_i)}$. Having determined the slack time of task v_i, the running frequency of v_i can be calculated by equations (11) and (12). At last, a push operation is used to ensure a more reasonable assignment. Algorithm 5 can deal with time slots appropriately, so it can reduce significant amount of energy consumption in a wide range of applications.

Algorithm 5 Slacking tasks

Require: $\Im'$, £, μ, P, G.

Ensure: $\Im$.

```
mark all tasks as unprocessed,
for ℑ′ is not null do
    popup the unprocessed task v_i with the largest finish time from ℑ′;   ▷ let p_k denote the assigned processor for v_i,
    calculate lft(v_i, p_k) by equation (6),
    calculate uslack(v_i) by equation (10),
    if uslack(v_i) > 0 then
        calculate the f_ideal(v_i) by equation (11),
        pick f_run(v_i) by equation (12),
        calculate T_exe(v_i) by equation (13),
    else
        set T_exe(v_i) = ω_i,k,
    end if
    update the start and finish time of task v_i,
    mark task v_i as processed,
end for
mark all tasks as unprocessed,
while there are unprocessed tasks do
    pick the unprocessed task v_i with the earliest start time,
    assign v_i on the mapped processor as early as possible,   ▷ push task v_i forward,
    update the start and finish time of task v_i,
    mark task v_i as processed,
end while
```

4.4 Time complexity of EALS

The time complexity of EALS is expressed in terms of the number of nodes $|V| = n$, and the number of processors $|P| = m$. The task mapping can be done in $O(m \times n^2)$. The complexity of Algorithm 3 is $O(m^2 \times n^2)$. The complexity of Algorithm 5 is bounded by $O(n^2)$. So the overall time complexity of EALS is $O(m^2 \times n^2)$.

5 Performance evaluation

In this section, we present the simulation results obtained from our EALS algorithm. We compare the EALS heuristic with two recently proposed algorithms, EES (Huang et al., 2012) and ETS (Wang et al., 2013). The EES algorithm slacks the room for the non-critical tasks and schedules the tasks nearby running on a uniform frequency for global optimality, and it still meets the performance-based SLA. The ETS algorithm evenly distributes the slack obtained by *makespan* extension to critical tasks, and then slacks the room for the non-critical tasks. For better comparison, we take the output of Algorithm 2 as the initial input for the three algorithms.

The performance is measured in terms of the total energy consumption. Here, we define a parameter energy-consumption-ratio (*ECR*) as the energy consumption metric:

$$ECR = \frac{E_{total}}{E_{total}(ees)}, \tag{14}$$

where $E_{total}(ees)$ is the total energy consumption of the EES algorithm, and E_{total} is the total energy of a compared algorithm. The *makespan* extension is determined by: $makespan \le (1 + \mu) \times makespan_{best}$, where $makespan_{best}$ is the schedule length of Algorithm 2. We present experimental results for *makespan* extension ratios equal to 0.00 (no extension), 0.05, 0.10, 0.15, 0.20 and 0.25, respectively.

The random generated application graphs and two real-world application graphs are utilised to evaluate the performance of the proposed algorithm. There are 500 random graphs generated for each scenario and we take the average *ECR* as final results in order to avoid scattering effects.

5.1 Randomly generated application graphs

The random generated application graphs include three fundamental characteristics:

1 n: the size of DAG

2 *CCR*: communication to computation cost ratio

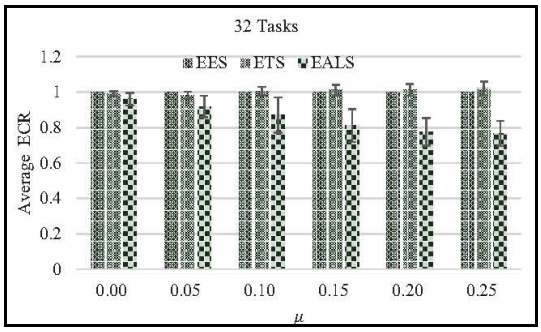
3 λ: parallelism factor (Tang et al., 2010; Mei et al., 2014).

In our experiments, the number of tasks in a DAG is selected from the *set* {32, 64, 100, 200, 400, 800}, the communication to computation cost ratio is determined by the *set* {0.2, 0.5, 1.0, 2.0, 5.0}, and the parallelism factor is picked from the *set* {0.2, 0.5, 1.0, 2.0, 5.0}.

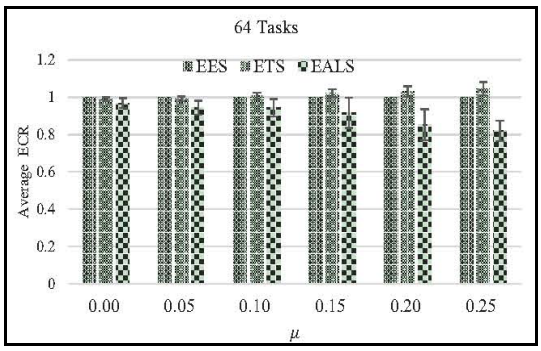
Figures 1 and 2 present the first two set of experiments compared to average ECR of the algorithms with respect to various number of tasks. Based on the observations from the two set of results, we can find that the proposed EALS

algorithm is able to decrease the total energy consumption with the determined performance constraint. The EALS algorithm obtains more energy saving with the increase of extension factor under the same configurations. For instance in Figure 1(a), the average ECR of EALS is less than the EES and ETS algorithms by: (3.92%, 3.05%), (8.11%, 6.54%), (12.96%, 13.47%), (18.91%, 19.84%), (22.35%, 23.34%), and (23.30%, 24.91%), for value of extension factor of 0.00, 0.05, 0.10, 0.15, 0.20 and 0.25, respectively. From Figure 2, we can see that the EALS approach consume less energy when the number of tasks is low. Since the EALS can power off some high energy-consumption processors guided by the heuristic strategy. As the number of tasks growth, there is only a few of processors can be powered off, so the total energy consumption is less distinct than otherwise. But the EALS heuristic still outperforms the EES and ETS algorithms. For instance in Figure 2, when the number of tasks is 800, the average ECR of our approach is less than the EES and ETS algorithms by: (1.65%, 1.38%), (1.03%, 2.04%), (3.66%, 6.35%), (9.29%, 13.29%), (7.85%, 13.24%), and (12.97%, 19.31%), for value of extension factor of 0.05, 0.10, 0.15, 0.20, and 0.25, respectively.

Figure 1 The average ECRs with respect to various numbers of tasks (CCR = 1.0, P = 4, λ = 1.0) (see online version for colours)

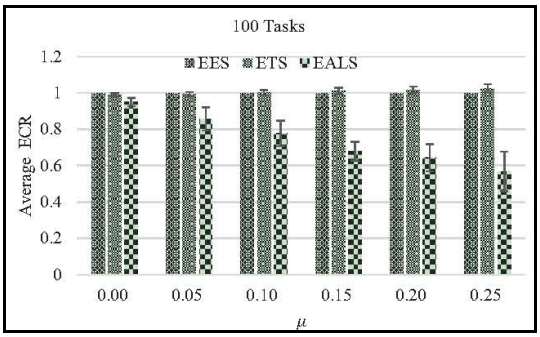


(a)

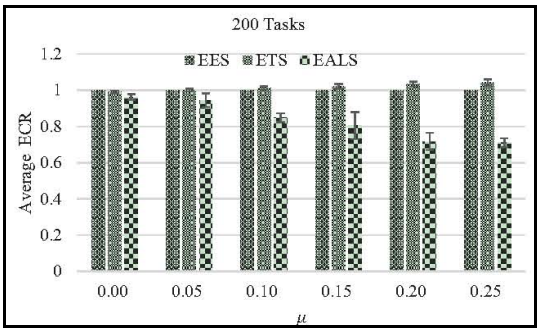


(b)

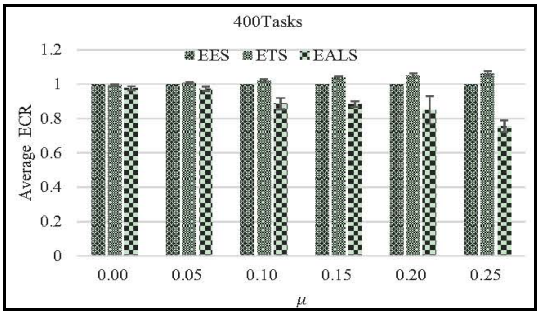
Figure 2 The average ECRs with respect to various numbers of tasks (CCR = 1.0, P = 9, λ = 1.0) (see online version for colours)



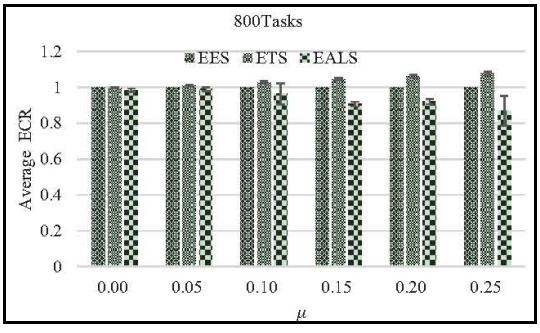
(a)



(b)



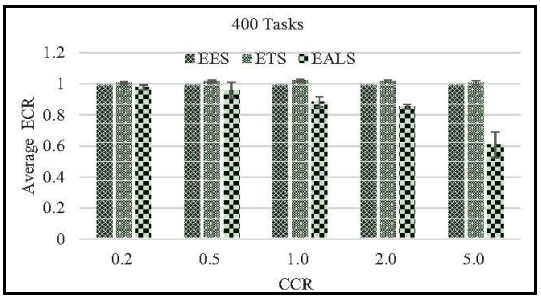
(c)



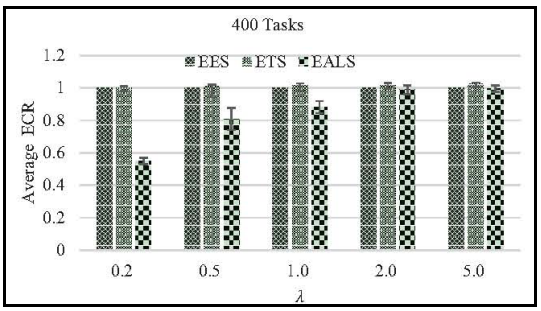
(d)

The third set of experiments compare the average ECR of the algorithms with respect to CCRs and parallelism factors, which is shown in Figure 3. In Figure 3, we can find that the performance of EALS algorithm improve with the increase of CCR, since a high CCR application can be considered as a communication-intensive application and our heuristic approach can assign tasks on a few of processors effectively that eliminates a lot of communication costs. A low parallelism factor λ leads to a deeper DAG, so the Algorithm 2 assign all tasks on a few processors. However, the EES and ETS algorithms do not consider the other underutilised processors. Instead, our approach can minimise the number of high power processors used and power off the idle processors. Consequently, the proposed algorithm can reduce more energy consumption in a system.

Figure 3 The average ECRs with respect to CCRs and parallelism factors ($\mu = 0.10$, $P = 9$) (see online version for colours)



(a)

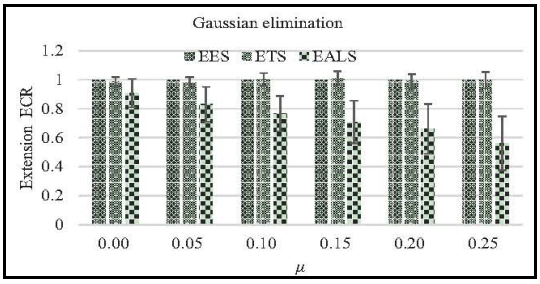


(b)

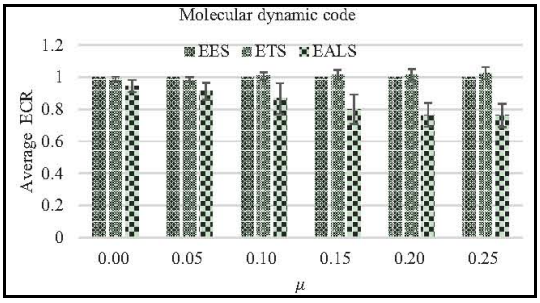
5.2 *Application graphs of real-world problems*

In this subsection, we consider application graphs of several real-world problems. The first two real-world applications are respectively the Gaussian elimination application (Cormen et al., 2009), which is a 5 × 5 matrix; and the molecular dynamic code (Kim and Browne, 1988), which consists of 41 tasks. The latter two real-world applications are respectively the fork-join (Yang and Gerasoulis, 1994), which consists of 18 tasks; and the partition algorithm (Li, 2012), which consists of 22 tasks. We compare the average ECRs of algorithms with various extension factors in Figures 4 and 5 respectively.

Figure 4 The average ECRs for the Gaussian elimination and molecular dynamic code ($CCR = 1.0$, $P = 4$) (see online version for colours)

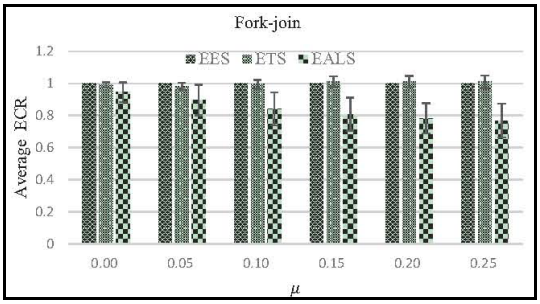


(a)

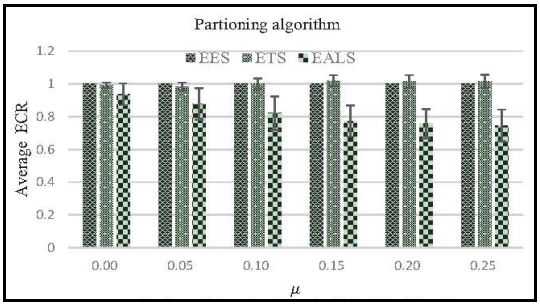


(b)

Figure 5 The average ECRs for the fork-join and partition algorithm ($CCR = 1.0$, $P = 4$), (a) fork-join (degree = 3, depth = 1 and width = 3) (b) a partition algorithm (width = 2 and height = 3) (see online version for colours)



(a)



(b)

From Figure 4, we can observe that the EALS approach consumes less energy than the EES and ETS algorithms, and the results are consistent with the results of randomly generated applications. With the increasing on extension factor, the energy-saving effect of EALS is increasingly obvious. For instance in Figure 4(a), the average ECR of the EALS approach is less than the EES and ETS algorithms by: (9.23%, 7.94%),(16.90%, 15.32%),(23.45%, 23.54%),(29.14%, 29.85%),(34.04%, 33.42%) and (43.96%, 43.81%), for value of extension factor of 0, 0.05, 0.10, 0.15, 0.20 and 0.25, respectively. We can find out that EALS comes to the best scene when the extension factor is equal to 0.25. Since the room for improvement with the increase in extension value, the EALS algorithm can deal with these tasks in an effective manner. The energy-saving effect of scheduling a Gaussian elimination application is superior to the effect of scheduling a molecular dynamic application, since the number of tasks in a Gaussian elimination application is lower than those of the counterpart, that makes more processors powered off by the proposed heuristic.

Figure 5 presents the average ECR of the last two real applications. The experimental results reveal that the proposed strategy can reduce significant amount of energy consumption compared with other algorithms. For instance in Figure 5(a), the average ECR of the proposed approach is less than the EES and ETS algorithms by: (5.50%, 4.55%), (10.45%, 8.82%), (15.73%, 15.25%), (19.24%, 19.96%), (21.80%, 22.55%) and (23.16%, 23.93%), for value of extension factor of 0.00, 0.05, 0.10, 0.15, 0.20 and 0.25, respectively. Meanwhile, the average ECR of several algorithms have the same trend in fork-join applications and partition applications. These results benefit most from the characteristic of heuristic scheduling strategy.

6 Conclusions and future work

Energy efficiency has become one of the most crucial research issues in cloud data centres. A good energy aware scheduling strategy is the key to reduce the total energy consumption in systems. In this paper, we propose a novel energy aware list-based scheduling algorithm for parallel applications in the context of SLA on DVS-enabled cloud data centres. We optimise the initial schedule by powering off part of processors and migrating some high energy-consumption tasks to an appropriate processor, then we take advantage of the DVS technique to further optimise the schedule. In order to prove the validity of the proposed EALS algorithm, we have performed a large number of experiments. For example, randomly generated graphs and several real-world applications are examined in our experiments. The experimental results show that EALS can reduce more energy consumption compared with two existing algorithms, and it still meets SLA.

In the further, the system reliability will be considered. We will optimise the EALS algorithm further and apply this algorithm to a Hadoop cloud system.

Acknowledgements

The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), and the National Natural Science Foundation of China (Grant Nos. 61370095, 61472124).

References

Bansal, S., Kumar, P. and Singh, K. (2003) 'An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems', *Parallel and Distributed Systems*, *IEEE Transactions on*, Vol. 14, No. 6, pp.533–544.

Bansal, S., Kumar, P. and Singh, K. (2005) 'Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs', *Journal of Parallel and Distributed Computing*, Vol. 65, No. 4, pp.479–491.

Benini, L., Bogliolo, A. and De Micheli, G. (2000) 'A survey of design techniques for system-level dynamic power management', *Very Large Scale Integration (VLSI) Systems*, *IEEE Transactions on*, Vol. 8, No. 3, pp.299–316.

Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C. (2009) *Introduction to algorithms*, MIT press. Dargie, W. (2012) 'Dynamic power management in wireless sensor networks: state-of-the-art', *Sensors Journal*, *IEEE*, Vol. 12, No. 5, pp.1518–1528.

Garey, M.R. and Johnson, D.S. (1979) 'Computers and intractability: a guide to the theory of np-completeness'.

Hagras, T. and Janeček, J. (2005) 'A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems', *Parallel Computing*, Vol. 31, No. 7, pp.653–670.

Huang, Q., Su, S., Li, J., Xu, P., Shuang, K. and Huang, X. (2012) 'Enhanced energy-efficient scheduling for parallel applications in cloud', in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012)*, IEEE Computer Society, pp.781–786.

Ilyas, M.U. and Khan, S.A. (2001) 'A clustering heuristic algorithm for scheduling periodic and deterministic tasks on a multiprocessor system', in *Multi Topic Conference, IEEE INMIC, Technology for the 21st Century, Proceedings, IEEE International*, IEEE, pp.1–5.

Kim, S.J. and Browne, J.C. (1988) 'A general approach to mapping of parallel computation upon multiprocessor architectures', in *International Conference on Parallel Processing*, Vol. 3, p.8.

Kwok, Y-K. and Ahmad, I. (1996) 'Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors', *Parallel and Distributed Systems*, *IEEE Transactions on*, Vol. 7, No. 5, pp.506–521.

Kwok, Y-K. and Ahmad, I. (1999) 'Static scheduling algorithms for allocating directed task graphs to multiprocessors', *ACM Computing Surveys (CSUR)*, Vol. 31, No. 4, pp.406–471.

Li, K. (2012) 'Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers', *Computers*, *IEEE Transactions on*, Vol. 61, No. 12, pp.1668–1681.

Liang, A., Xiao, L. and Ruan, L. (2013) 'Adaptive workload driven dynamic power management for high performance computing clusters', *Computers & Electrical Engineering*, Vol. 39, No. 7, pp.2357–2368.

Comment [t2]: Author: Please provide the journal title where the paper/article was taken.

Author: Please provide the volume number, issue number and page numbers.

Mehta, N. and Amrutur, B. (2012) 'Dynamic supply and threshold voltage scaling for CMOs digital circuits using in-situ power monitor', *Very Large Scale Integration (VLSI) Systems*, *IEEE Transactions on*, Vol. 20, No. 5, pp.892– 901.

Mei, J. and Li, K. (2012) 'Energy-aware scheduling algorithm with duplication on heterogeneous computing systems', in *Grid Computing (GRID, ACM/IEEE 13th International Conference on*, IEEE, pp.122–129.

Mei, J., Li, K. and Li, K. (2014) 'A resource-aware scheduling algorithm with reduced task duplication on heterogeneous computing systems', *The Journal of Supercomputing*, Vol.68, No.3, pp.1347–1377.

Mittal, S. (2014) 'A survey of techniques for improving energy efficiency in embedded computing systems', *International Journal of Computer Aided Engineering and Technology*, Vol. 6, No. 4, pp.440–459.

Niu, L. and Quan, G. (2013) 'Leakage-aware scheduling for embedded real-time systems with (*m*, *k*)-constraints', *International Journal of Embedded Systems*, Vol. 5, No. 4, pp.189–207.

Piyatamrong, B., Ohara, S. and Kantakajorn, S. (2000) 'GTCs: a greedy task clustering and scheduling algorithm for distributed memory processor architecture', in *High Performance Computing in the Asia-Pacific Region, Proceedings, The Fourth International Conference/Exhibition on*, IEEE, Vol. 1, pp.310–314.

Quan, D.M. (2007) 'Error recovery mechanism for grid-based workflow within SLA context', *International Journal of High Performance Computing and Networking*, Vol. 5, No. 1, pp.110–121.

Shao, Z., Zhuge, Q., Zhang, Y. and Sha, E.H. (2004) 'Algorithms and analysis of scheduling for low power high-performance DSP on VLIW processors', *International Journal of High Performance Computing and Networking*, Vol. 1, No. 1, pp.4–16.

Tang, X., Li, K., Liao, G. and Li, R. (2010) 'List scheduling with duplication for heterogeneous computing systems', *Journal of Parallel and Distributed Computing*, Vol. 70, No. 4, pp.323–329.

Terzopoulos, G. and Karatza, H.D. (2013) 'Dynamic voltage scaling scheduling on power-aware clusters under power constraints', in *Distributed Simulation and Real Time Applications (DS-RT, IEEE/ACM 17th International Symposium on*, IEEE, pp.72–78.

Topcuoglu, H., Hariri, S. and Wu, M-y. (2002) 'Performance-effective and low-complexity task scheduling for heterogeneous computing', *Parallel and Distributed Systems*, *IEEE Transactions on*, Vol. 13, No. 3, pp.260–274.

Wang, L., Khan, S.U., Chen, D., Ko lodziej, J., Ranjan, R., Xu, C-z. and Zomaya, A. (2013) 'Energyaware parallel task scheduling in a cluster', *Future Generation Computer Systems*, Vol. 29, No. 7, pp.1661–1670.

Wu, C-M., Chang, R-S. and Chan, H-Y. (2014) 'A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters', *Future Generation Computer Systems*, Vol. 37, pp.141–147.

Wu, J. and Wu, J-X. (2014) 'An SRP-based energy-efficient scheduling algorithm for dependent real-time tasks', *International Journal of Embedded Systems*, Vol. 6, No. 4, pp.335–350.

Wu, M-Y. and Gajski, D.D. (1990) 'Hypertool: a programming aid for message-passing systems', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 3, pp.330–343.

Xiao, P. and Han, N. (2014) 'A novel power-conscious scheduling algorithm for data-intensive precedence constrained applications in cloud environments', *International Journal of High Performance Computing and Networking*, Vol. 7, No. 4, pp.299–306.

Yang, T. and Gerasoulis, A. (1994) 'DSC: scheduling parallel tasks on an unbounded number of processors', *Parallel and Distributed Systems*, *IEEE Transactions on*, Vol. 5, No. 9, pp.951–967.

Zhong, X. and Xu, C-Z. (2007) 'Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee', *Computers*, *IEEE Transactions on*, Vol. 56, No. 3, pp.358–372.