Journal of Intelligent & Fuzzy Systems 36 (2019) 5135–5152
DOI:10.3233/JIFS-171927
IOS Press

EATSDCD: A green energy-aware scheduling algorithm for parallel task-based application using clustering, duplication and DVFS technique in cloud datacenters

Behnam Barzegar[a], Homayun Motameni[a,*] and Ali Movaghar[b]
[a]*Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran*
[b]*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*

Abstract. Energy consumption and performance metrics have become critical issues for scheduling parallel task-based applications in high-performance computing systems such as cloud datacenters. The duplication and clustering strategy, as well as Dynamic Voltage Frequency Scaling (DVFS) technique, have separately been concentrated on reducing energy consumption and optimizing performance parameters such as throughput and makespan. In this paper, a dual-phase algorithm called EATSDCD which is an energy efficient time aware has been proposed. The algorithm uses the combination of duplication and clustering strategies to schedule the precedence-constrained task graph on datacenter processors through DVFS. The first phase focuses on a smart combination of duplication and clustering strategy to reduce makespan and energy consumed by processors in an effort to execute Directed Acyclic Graph (DAG) while satisfying the throughput constraint. The main idea behind EATSDCD intended to minimize energy consumption in the second phase. After determining the critical path and specifying a set of dependent tasks in non-critical paths, the slack time for each task in non-critical paths was distributed among all dependent tasks in that path. Then, the frequency of DVFS-enabled processors is scaled down to execute non-critical tasks as well as idle and communication phases, without extending the execution time of tasks. Finally, a testbed is developed and different parameters are tested on the randomly generated DAG to evaluate and illustrate the effectiveness of EATSDCD. It was also compared against duplication and clustering-based algorithms and DVFS-based algorithms. In terms of energy consumption and makespan, the results show that our proposed algorithm can save up to 8.3% and 20% energy compared against Power Aware List-based Scheduling (PALS) and Power Aware Task Clustering (PATC) algorithms, respectively. Furthermore, there is 16% improvement over Parallel Pipeline Latency Optimization (PaPilo) algorithm with $En_{cur} = 1.2En_{min}(G)$. In comparison with Reliability Aware Scheduling with Duplication (RASD) algorithm, the execution time has been reduced in heterogeneous environments.

Keywords: Green computing, cloud data centers, dynamic voltage and frequency scaling (DVFS), task duplication, energy consumption, slack time, throughput

1. Introduction

Nowadays, energy consumption has become a critical issue in high performance distributed computing systems (HPDCSs). Therefore, green computing attempts to minimize energy consumption, carbon

*Corresponding author. Homayun Motameni, Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran. Tel./Fax: +98 9111140554; E-mail: motameni@iausari.ac.ir.

footprint and CO_2 emissions in HPDCSs, including clusters, grids and clouds made up of a large number of parallel processors [1].

Recent studies suggest that nearly 1.5–2% of total energy worldwide is consumed by datacenters. Such tremendous growth can be attributed to popularity of distributed computing platforms such as clusters, grids and clouds. Moreover, previous studies indicated that about 52% of energy in datacenters is consumed by computing systems, while the rest is consumed by support systems. In fact, it has been estimated that the electricity consumed in the American datacenters will expand from 91 billion kWh in 2013 to roughly 140 million kWh in 2020 [2].

Hence, it is crucial to schedule precedence-constrained parallel applications, one of the models applied in science and engineering fields, on homogeneous and heterogeneous computing systems like cloud computing infrastructures with regard to energy consumption and other performance parameters [3, 4]. Scheduling is also considered a well-known NP-Hard optimization problem [5], for which numerous heuristic algorithms have so far been proposed [6, 7].

The data analysis steps can be expressed as DAG, which operates on a stream of input data-task in DAG repeatedly receiving input data items from their predecessors, while writing the output to their successors. Makespan and throughput are typical performance-related metrics to measure the performance of a DAG (precedence-constrained parallel application). Makespan is the maximum time to process an individual data item, in which the task in the DAG has been completed [4], while throughput simply counts the number of tasks completed over the makespan [8].

Hence, it is essential to create a compromise between performance and energy consumption, thereby to decrease makespan and energy consumption while increasing throughput. Green computing is therefore crucial for ensuring the future growth of cloud computing will be persistent. The design and development of green software for scheduling precedence-constrained parallel applications can directly affect performance parameters as well as energy consumed by processors and communication networks in cloud datacenters.

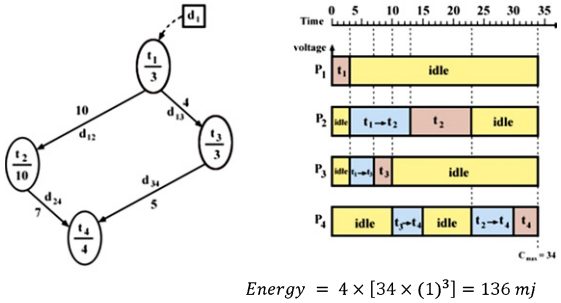
Our objective in this paper was to propose an energy-efficient, time-aware, scheduling heuristic strategy called EATSDCD for energy-aware task duplication-based scheduling algorithm of parallel tasks on cloud datacenters. In order to achieve good performance and energy consumption for a given parallel application, we proposed a novel task scheduling algorithm based on clustering and duplication design pattern and dynamic voltage frequency scaling (DVFS) technique. The proposed algorithm aims to reduce the communication energy through task duplication and clustering. However, these duplicate-based scheduling strategies replicate tasks and clustering by another task only according to the energy difference between current task computation energy and communication energy of these two tasks. We have developed an application which can be represented as a DAG.

This application involves four tasks called t_1, t_2, t_3 and t_4, the execution timed of which are 3, 10, 3 and 4 time units with four communication links called d_{12}, d_{13}, d_{24} and d_{34}, the communication times of which are 10, 4, 7 and 5, respectively.

Figure 1 illustrates an example of processor allocation, and the values of makespan, throughput and energy consumption, without using duplication, clustering and DVFS technique. Figure 2 provides a DAG example using clustering and DVFS technique, while Fig. 3 displays a DAG example using clustering, duplication and DVFS technique.

Task clustering is a technique to minimize and eliminate the expensive communication cost during data transfer between tasks through tasks allocation to same processors. In practice, cluster refers to a set of tasks executed on an identical processor. Applied correctly, this technique can mitigate makespan and energy consumption, maximize throughput and minimize the number of active processors for task scheduling [50].

Task duplication is a technique that causes to prevent the communication cost between processors assigned to the tasks, which are in communication, by creating data locality. The data locality is generated



$$Energy = 4 \times [34 \times (1)^3] = 136\,mj$$
$$makespan = 34$$

Fig. 1. A DAG scheduling example without using clustering, duplication and DVFS.

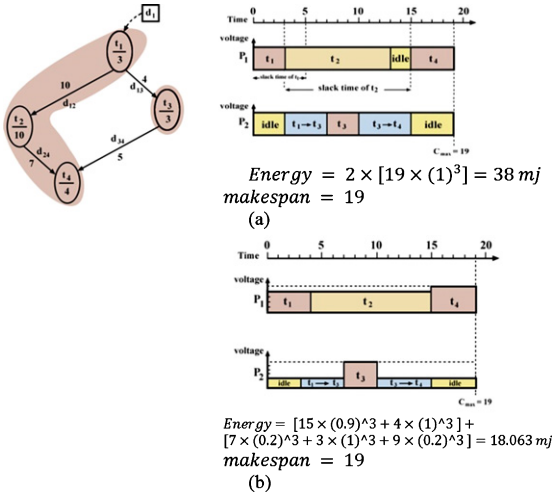


Fig. 2. (a) Gantt chart for P_1 and P_2 after clustering, (b) Energy Gantt chart after stack time distribution (employ DVFS).

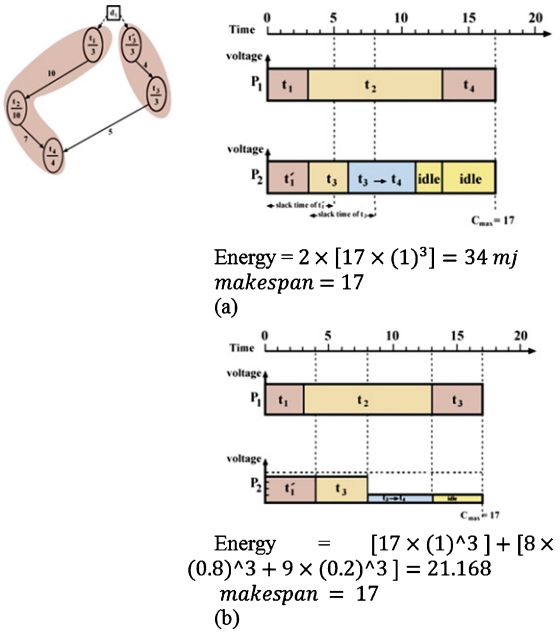


Fig. 3. (a) Gantt chart for p_1 and p_2 after duplication & clustering, (b) Energy Gantt chart after stack time distribution (employ DVFS).

by replicating and copy of specific tasks on multiple processors. In fact, this technique prevents data transfer between predecessor and successor, thereby to reduce communication costs. This technique can be far more effective in reduction of makespan and energy consumption for DAG with greater CCRs [6, 9–13].

DVFS technique, modern processors are equipped with dynamic voltage frequency scaling (DVFS) technique, which reduces energy consumption by switching between processor's voltage and frequency pairs to execute tasks during slack times and idle or communication phases [14]. The processor dispatch strategy of tasks is assumed to allocate each cluster on an independent processor.

As can be seen in Fig. 2, t_1 has been clustered with t_2 and t_4 so as to avoid their expensive communication link. This in turn mitigates energy consumption and makespan, while enhancing throughput. Duplication can reduce communication costs by allocating the copies of tasks to extra processors. Figure 3 shows that t_1 is duplicated and simultaneously allocated to two processors. This can hide the communication cost of d_13. Compared to clustering, the combination of duplication and clustering techniques has delivered better results. After applying the clustering and duplication techniques on the primary DAG sample, the results of the two strategies can be seen in Figs. 2 and 3, where DVFS should be adopted. To that end, the critical path was first determined and slack times of non-critical paths were calculated. Then, the idle and communication phases were specified by reducing the voltage and frequency of processors through the DVFS technique, which significantly reduces energy consumption. Given the fact that dynamic power consumption of processors has been calculated by Equation $P = ACv^2 f$, and the values of A and C are constant for each processor in addition to $v\alpha f$ consequently $P\alpha f^3$. Given that $= P \times t$, therefore $En\alpha f^3 \times t$ will be true [29]. At this stage, we can calculate the amount of energy consumed by processors by executing the given graph sample.

The rest of the paper has been organized as follows. In Section 2, we present the related work and the current state-of-the-art in energy-aware scheduling based on DVFS and duplication technique. System model including an architecture model, parallel task model, resource model, DVFS model and the multi-objective estimation model are illustrated in Section 3. In Section 4, we introduce our new EATSDCD algorithm for solving the problem. This algorithm includes two phases called EATSDC and EADVFSA. The time complexity analysis for the proposed algorithm and the tracing EATSDCD algorithm on given DAG are presented in Sections 5 and 6, respectively. Section 7 explores the performance constraint setting, randomly generated DAG and experimental results compared with other algorithms. Finally, Section 8 concludes this paper and plans for future work.

2. Related work

Traditional algorithms mainly focused on scheduling of precedence-constrained parallel applications on distributed platforms such as clusters, grids, and clouds, minimizing the total completion time or makespan without worrying about the energy consumed in datacenters [15, 16]. As the Information and Communication Technology (ICT) has developed over the last few years, there have been a growing number of datacenters and accordingly a dramatic increase in energy consumption. This has subsequently left negative impact on the environment through generation of greenhouse gases and excessive emission of CO_2 [49]. In recent years, great efforts have been made to mitigate the energy consumed by processors at datacenters using 1) DVFS techniques [17–21, 53], 2) changing the scheduling policies for allocating tasks on available processors [21], 3) dynamic power management (DPM) [22], 4) Working Vacation [23–25] and 5) redesign of algorithms using energy-efficient pattern in compilers [26]. These efforts have replied on several design patterns such as clustering and duplication. This section will discuss relevant studies previously conducted on a few conventional techniques.

2.1. Energy reduction based on DVFS technique

Dynamic voltage and frequency scaling (DVFS) has been recognized as an effective technique to reduce energy consumption of processors through simultaneous minimization of frequency and supply voltage for slack time slot of tasks as well as communication and idle phases.

The authors in [20] employed an energy-aware scheduling heuristic algorithm called PALS and PATC to simultaneously reduce makespan and energy consumption for scheduling parallel tasks in a cluster through DVFS technique. After determining the critical path and non-critical paths, the proposed algorithm assigns jobs in the critical paths to processors with the highest voltage/frequency. Then, the slack time of each jobs is calculated in the non-critical paths, and the voltage/frequency of the assigned processors is scaled down to process the non-critical jobs. This strategy mitigates energy consumption without increasing makespan. By negotiating with users, based on the Green service-level agreement (SLA) negotiation, a compromise is made between further reducing energy consumption and thus increasing makespan. Another approach to scheduling tasks has been proposed to reduce energy consumption using the DVFS technique [27]. This technique has been adopted to dynamically control the frequency and voltage of cloud computing servers.

The scheduling algorithm takes into account the maximum job (F_{max}) and minimum job (F_{min}) frequencies given to each job and multiple server Si running at maximum Si (F_{max}) and minimum Si (F_{min}) frequencies. For specific jobs, the scheduling algorithm efficiently assigns proper servers that run between (F_{min}, F_{max}) to jobs according to requirements of job frequencies.

Juarez et al. [26] proposed a real-time dynamic scheduling method called Multi-heuristic Resource Allocation (MHRA) for efficient execution of task-based applications on a distributed computing platform of cloud computing. This served to mitigate energy consumption and makespan. This method involved a polynomial time algorithm combining a set of heuristic rules and resource allocation techniques. In order to balance the two-objective function, a weight factor was introduced $\alpha\, 0 \leq \alpha \leq 1$, by which the user can specify the significance of each objective.

Yikun Hu et al. [19] developed an algorithm called Energy Aware Service Level Agreement (EASLA) for scheduling parallel applications through DVFS technique, while maintaining the SLA on a cluster platform.

The main idea behind EASLA algorithm is to allocate each slack to a maximum set of independent tasks for each task using a compatible task matrix and scale frequencies down to minimize energy consumption within certain extension rate of makespan mutually accepted by user and service provider.

Furthermore, Mezmaz M et al proposed a hybrid, parallel, multi-objective genetic algorithm to solve the problem of scheduling parallel precedence-constrained applications in an effort to simultaneously mitigate the overall execution time of tasks and energy consumed in cloud computing. The energy storage involved DVFS, where each processor can operate at different clock frequencies. This approach has been evaluated with the Earliest Finish Time (FFT) task graph, which is a real-world application [3].

Cloud computing offers utility-oriented IT services to consumers based on pay-as-you-go model. This model involves a payment method for services charging based on usage only for resources needed [47].

Datacenters have extensively grown to provide service to clients globally. Hence, the datacenter hosts consume a huge amount of power for

Infrastructure as a Service (IaaS), Software as a service (SaaS), Platform as a Service (Paas) applications. This consequently leaves an adverse impact on the natural environment. Beloglazov A et al. proposed an architectural framework for energy-aware, heuristic allocation of data center resources to consumer applications, while considering the quality of services (QOS) and power usage characteristics of the devices [28].

Reducing energy consumption in an idle servers and running a server with CPU utilization controlled by DVFS techniques, and authors' approach has been evaluated through CloudSim toolkit.

The authors in [29] proposed a new task slack time algorithm for task scheduling in distributed computing systems using DVFS technique.

In [30], a scheduling algorithm called Energy Aware DAG Scheduling (EADAGS) was developed on heterogeneous distributed processor system using dynamic voltage scaling (DVS) with decisive path scheduling (DPS) to achieve minimal finish time and energy consumption.

In [53], the problem of scheduling precedence-constrained parallel applications on multiprocessor system was proposed to increase throughput and minimize energy consumption by dynamic voltage scaling.

H.K Imura et al. [31] introduced an algorithm reclaims slack time, where slack time in parallel applications was executed on a power-scalable cluster computing using DVFS. Moreover, the newly proposed method was evaluated by a toolkit called Powerwat, which includes a monitoring and control tool.

Ding et al. [14] proposed an energy consumption optimization algorithm known as Energy Efficient virtual machines scheduling (EEVS) for scheduling of virtual machines given the deadline constraint using DVFS.

Shu et al. have offered other examples of how to optimize resource allocation using an improved clonal selection algorithm with bi-objective criteria in cloud computing. The authors proposed an improved clonal selection algorithm (ICSA) based on makespan optimization and improvement of energy efficiency in datacenters, capable of effectively meeting the SLA requested by consumers [55].

Table 1
Previous studies on task duplication technique

Reference	Target platform	Scheduling objective
TDGA [11]	Homo/heter	Schedule length Load balancing satisfaction
RASD [33]	Heter	Reliability Makespan
CA-D [6]	Homo/heter	Speed up Energy consumption
AES [34]	Homo	Performance Schedule length
EPTAC [2]	Homo	Energy consumption CPU utilization
ASA [35]	Homo	Makespan
EAMD [22]	Heter	Energy consumption Cost Reliability Makespan
CPFD [36], PY [37], LWB [38], BTDH [39], DSH [40]	Homo/heter	Efficiency Cost Normalized scheduling length
TCLO [8]	Homo	Latency Throughput Power consumption
SDS [54]	Homo	Schedule length Number of processors
NEADS [43]	Homo	Makespan Energy consumption
PaPIio [41]	Homo	Latency Throughput Power

2.2. Energy reduction based on scheduling policy and duplicate technique

Table 1 summarizes the previous studies on scheduling policies with duplicate technique for allocating tasks on processors with different objectives. We presented algorithms in each reference, while target platforms can be classified to homogeneous and heterogeneous environments and scheduling objectives.

3. System model

In this section, formal definition for system architecture model, parallel task model, DVFS model, resource model, energy consumption model in processors as well as interconnections, performance model under some assumption and restrictions, which are employed in problem formulation has been proposed. Table 2 summarizes the notations used in this paper.

3.1. Architecture model

This section introduces our proposed architecture model for the parallel task scheduling environment on cloud datacenters. The architecture model illustrated in Fig. 4 comprises three layers each including different sections.

3.1.1. COMP Superscalar layer

COMP Superscalar (COMPSs) layer is a framework aiming to ease the development and execution of task-based applications for distributed infrastructure, such as clusters, grids and clouds, and a runtime system which manages several execution aspects of applications. Besides, it keeps the underlying infrastructure transport to the application [26].

3.1.2. Datacenter resource layer

This layer contains several computational nodes, each including multiple virtual machines. Each virtual machine includes multiple processors, disks, memories, and communication networks. Processors are DVFS-enabled and are assigned to execute tasks.

3.2. Parallel task model

The sequential program sent by the user to COMPSs is converted into DAG by the task dependency analyzer component. The created DAG, called task dependency graph, is displayed as $G(T, D)$, where:

Table 2
Definition of notations

Notation	Definition
t_i	The task number ith
N	The number of tasks (nodes) in DAG
w_i	The weight of task ith
t_i^{st}	The start time of task ith
$et(t_i, p_j)$	The execution time of task ith on processor jth
t_i^{end}	The end time of task ith
CPI	The clock per instruction
$succ(t_i)$	The set of successors of task ith
$pred(t_i)$	The set of predecessors of task ith
d_{ij}	The independent between task t_i and t_j
$ct(d_{ij})$	The communication time to transfer message d_{ij}
$et(C_i)$	The execution time of cluster c_i
$\|j\|$	The number of computational nodes
$\|k\|$	The number of VM in each computing node
$\|m\|$	The number of processors in each VM
$(\mathbf{v_j}, \mathbf{f_j})\|$	The voltage and frequency pairs of processor jth
$(\mathbf{v_{kj}}, \mathbf{f_{kj}})$	The voltage/frequency pairs of processor jth at level k
v_{highj}	The highest voltage of processor jth
f_{highj}	The highest frequency of processor jth
$p_j.f_k^{op}$	The processor jth operating frequency at level k
ND_{ti}	The number of duplication task ith
CCR	Communication to Computation Ratio
l	Communication link
p	Power consumption
$P_{dynamic}$	Dynamic Power consumption
P_{static}	Static Power consumption
E	Energy consumption
ec_{ij}	Communication energy by edge d_{ij}
PC	Power of interconnect
$C_{\max}$	Makespan (total length of the schedule)
CPL	Critical Path Length
$CommR(G)$	Communication Rate for DAG
$CompR(G)$	computation Rate for DAG
$Th(G)$	Throughout for DAG

- T: consists of a set of tasks in G, which can be represented by Equation 1. All tasks $\forall t_i \in T$ are the components of the application code (nodes in a DAG). These tasks are scheduled to run over different processors in the systems.

$$\mathrm{T} = \cup\{\mathrm{t_i}\},\ 1 \le i \le \mathrm{n} \tag{1}$$

Where

- $t_i$is a task ith in DAG.
- n is the total number of tasks.
- w_i is weight on task, t_i represents the instruction number of task t_i.
- t_i^{st} is the start time of task t_i.
- $et(t_i, p_j)$ is execution/computation time of task t_i on processor p_j, which is indivisible and its exe-

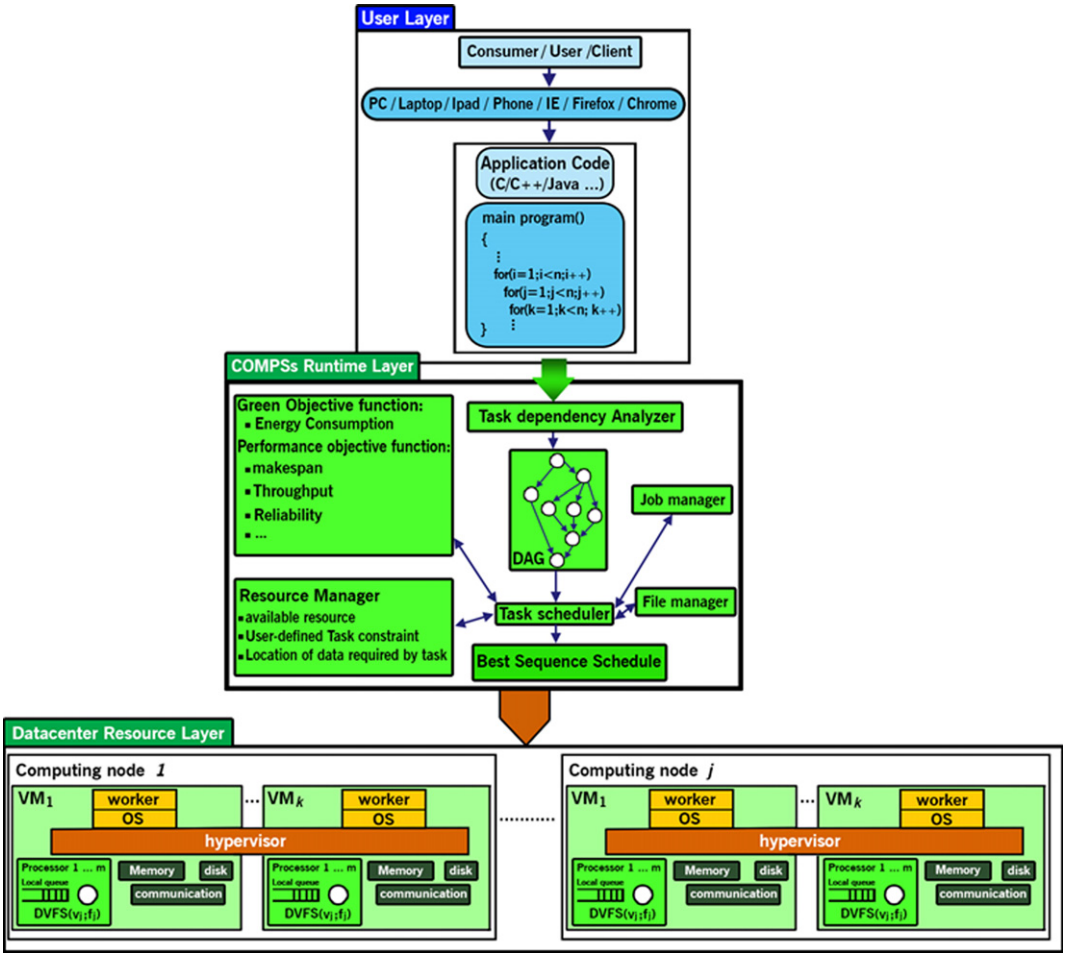


Fig. 4. Comprises three layers each including different sections.

cution cannot be interrupted. The task execution time t_i is calculated as indicated on Equation 2.

$$\mathrm{et}(\mathrm{t_i}, \mathrm{p_j}) = \frac{\mathrm{t_i.w} \times \mathrm{CPI}}{\mathrm{p_j.f^{op}}} \tag{2}$$

CPI denotes the number of clock cycle per instruction for a task by processor. The ideal CPI is 1.

- $\mathrm{t_i^{end}}$ is the end/finish time of task $\mathrm{t_i}$ calculated through Equation 3.

$$(\mathrm{t_i^{end}},\ \mathrm{p_j}) = (\mathrm{t_i^{st}},\ \mathrm{p_j}) + \mathrm{et}(\mathrm{t_i},\ \mathrm{p_j}) \tag{3}$$

- ***D***: consists of a set of directed edges between the tasks in G to represent precedence constraints of an edge, $d_{ij} \in D$ represents that task t_j is independent on task t_i, where t_j must be executed after end of t_i., t_i is the parent and t_j is the child. (*or t_i is the predecessor of t_j and t_j is the successor of t_i*). A task may have one or more inputs. When all inputs are available, the task is triggered to execute. After the execution, the task generates its output. In the DAG, we use $succ(t_i)$ to denote the set of successors of task t_i and $pred(t_i)$ to denote the set of predecessors of task t_i. A task with no predecessors, $pred(t_i) = \phi$, is called an entry task (t_{entry}) and a task with no successors, $succ(t_i) = \phi$ is called an exit task (t_{exit}). We require a single entry task and single exit task for a DAG. Since a given graph contains more than one entry or exit task, we can produce a new graph by connecting all entry tasks to a new zero-cost entry task or all exit tasks to a new zero-cost exit task. The communication costs between the tasks are zero.
- $ct(d_{ij})$ is communication time/cost of an edge d_{ij}, for transfer message d_{ij}, this time/cost is incurred if t_i and t_j are scheduled on different processors and is considered to be zero if t_i and

t_j are scheduled on the same processor.

3.3. Cloud datacenter resource model

We model the cloud datacenter resource layer as $CDR = \{j, k, m, l\}$, where:

- j represents the set of computational nodes, while there is a $|j|$ computing node in CDR
- k represents the set of virtual machines, while there is a $|k|$ virtual machine in each computing node.
- m represents the set of processors, while there is a $|m|$ processor in each virtual machine.
- The resource layer consists of multiple computing nodes $CN = \{cn_1, cn_2, \ldots, cn_j\}$, where node j represented by cn_j, each computing node cn_j consists of a set of virtual machines $V_j = \{V_{j1}, V_{j2}, \ldots, V_{jk}\}$, where virtual machine k of computing node j is represented by V_{jk}. Each virtual machine V_{jk} has a set of processors $C_{jkm} = \{C_{jk1}, C_{jk2}, \ldots, C_{jkm}\}$.
- l; all computing nodes, virtual machines and processors are fully interconnected with the same communication link l.

3.4. DVFS model

Nowadays, DVFS-enabled processors are employed to mitigate energy consumption in HPC systems [42].

DVFS-enabled processors can execute tasks during slack times as well as idle and communication phases using a discrete set of voltage and frequency pairs, $(v_i f_j)$. Assume that each processor has k DVFS levels in other words k processing operating points. Hence, supply voltage and frequency processor j can be described by Equation 4.

$$(\mathrm{v_i},\ \mathrm{f_j}) = \left\{ \begin{array}{l} (\mathrm{v_{lowj}}, \mathrm{F_{lowj}}) = (\mathrm{v_{1j}}, \mathrm{f_{1j}}) < (\mathrm{v_{2j}},\ \mathrm{f_{2j}}) < \cdots < \\ \quad (\mathrm{v_{kj}},\ \mathrm{f_{kj}}) = (\mathrm{v_{highj}},\ \mathrm{f_{highj}}) \end{array} \right\} \tag{4}$$

Where (v_{kj}, f_{kj}) is the voltage and frequency for processor j at level k.

Furthermore, the execution time of task t_i on processor p_j with the set of working frequencies from f_{1j} to f_{kj} can be calculated through Equation 5. In fact, the greater the frequency level the shorter the execution time.

$$\mathrm{et'(t_i, p_j)} = \left[\frac{\mathrm{t_i}.w \times \mathrm{CPI}}{\mathrm{p_j}.\mathrm{f}_1^{op}}, \frac{\mathrm{t_i.w} \times \mathrm{CPI}}{\mathrm{p_j}.\mathrm{f}_2^{op}}, \ldots, \frac{\mathrm{t_i.w} \times \mathrm{CPI}}{\mathrm{p_j}.\mathrm{f}_\mathrm{k}^{op}} \right] \tag{5}$$

3.5. Estimation model of DAG schedule

In this section, a few models are adopted by evaluation of DAGs at different sizes to estimate throughput, makespan and energy consumption.

3.5.1. Makespan estimation

Makespan ($C_{\max}$) is defined as the amount of time, from start to end for completing a set of sequences. The best effort in a scheduling algorithm is to minimize the maximum completion time (makespan). Equation 6 describes how the makespan of a DAG is calculated

$$\mathrm{C_{max}} = \left[\max(\text{task i.t}^{\mathrm{end}}) - \min(\text{task } j.\ \mathrm{t}^{st}) \right], \quad 1 \le \mathrm{i},\ \mathrm{j} \le \mathrm{n} \tag{6}$$

Critical Path (CP) of a DAG is the longest path from the entry node to the exit node in the graph. The lower bound of a schedule length is the minimum critical path length (CPMIN). If any task on the critical path is late, the tasks scheduling in graph is late. Since the number of processing sources for tasks execution is unlimited, $C_{\max}$ can be considered equal to the length of critical path for G graph.

$$\mathrm{C_{max}(G)} = \mathrm{CPL(G)} \tag{7}$$

3.5.2. Throughput estimation

The proposed model for evaluation of throughput has been adopted from [41]. It is essential to include communication rate (CommR(G)) and computation rate (CompR(G)) for the given DAG, G.

Since all tasks in a cluster such as C_i should be executed on one processor, the data elements are processed sequentially. The computation rate for cluster C_i is equal to $\frac{1}{et(C_i)}$ data items per unit of time. Given that the computation rate for the given DAG, G, is determined by the slowest cluster, we can calculate the CompR(G) as indicated by Equation 8

$$\mathrm{CompR(G)} = \min_{\forall \mathrm{C_i} \in \mathrm{C}} \& \frac{1}{\mathrm{et(C_i)}} \tag{8}$$

Moreover, the communication rate for each edge d_{ij} is equal to $\frac{1}{ct(d_{ij})}$ data items per unit of time. Given that the communication rate for G given DAG is determined by the lowest communication rate for all edges in G, the CommR(G) of the DAG can be calculated according to Equation 9.

$$\mathrm{CommR(G)} = \min_{\forall \mathrm{d_{ij}} \in \mathrm{D}}, \frac{1}{\mathrm{ct(d_{ij})}} \tag{9}$$

Since the computation and communication for different processors can be carried out simultaneously, the overall throughout can be calculated through Equation 10.

$$\mathrm{Th(G)} = \min_{\forall \mathrm{d_{ij}} \in \mathrm{D}} \{\mathrm{compR(G)}, \mathrm{commR(G)}\} \tag{10}$$

3.5.3. Energy consumption model

The energy consumed for scheduling of dependent tasks in computational systems is equal to total computation energy of processors for task execution as well as energy consumed to transfer data between processors across communication networks.

3.5.3.1. Computation energy

Nowadays, most processors are constructed using CMOS circuits. In such processors, power consumption is divided into two parts (dynamic power consumption and static power consumption) which are obtained through Equation 11.

Static power consumption, i.e. the main source of static current, is leakage current and reverse based PN junction when there is no circuit activity, whereas dynamic power consumption involves charging and discharging of capacitances when inputs are active [20, 32].

$$\mathrm{P} = \mathrm{P_{dynamic}} + \mathrm{P_{static}} \tag{11}$$

Given that the total energy consumed to execute parallel tasks involves computation energy by processors and communication energy between processors, the static part of power consumption can be ignored.

The dynamic power consumption of processors can be calculated through Equation 12 [52].

$$P_{dynamic} = ACv^2 f \tag{12}$$

Where A is the percentage of active logic gates, C is the effective load capacitance, v is the supply voltage and f is the frequency of processor.

Given that modern processors are equipped with DVFS technology, the maximum power consumption of processor $P_{proc.highest}$ occurs when it operates at maximum voltage $v_{highest}$ and frequency $f_{highest}$. Therefore, it can be concluded that the active power consumption for a processor under the voltage and frequency set (v_j, f_j) is calculated through Equation 13.

$$\begin{aligned} \mathrm{P_{procj}} &= \mathrm{P_{proc.highest}} \times \frac{\mathrm{v_j^2 \times f_j}}{\mathrm{v_{highest}^2 \times f_{highest}}} \\ \mathrm{P_{proc.highest}} &= ACv_{highest}^2 f_{highest} \end{aligned} \tag{13}$$

Since the proposed algorithm adopts the task duplication strategy for scheduling a DAG with n tasks on DVFS-enabled processors, the total energy consumption can be calculated through Equation 14.

$$\begin{aligned} \mathrm{P_{processor.active}} &= \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{P_{proc.higest}} \\ &\left(\sum_{\mathrm{j=1}}^{\mathrm{k}} \frac{\mathrm{v_j^2 \times f_j^2}}{\mathrm{v_{highest} \times f_{highest}}} + \mathrm{ND\_ti} \right) \\ \mathrm{E_{processors.active}} &= \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{P_{proc.higest}} \end{aligned}$$

$$\begin{aligned} &\left(\sum_{\mathrm{j=1}}^{\mathrm{k}} \frac{\mathrm{v_j^2 \times f_j}}{\mathrm{v_{highest}^2 \times f_{highest}}} \times \mathrm{et(t_i, p_m(v_j, f_j)} \right) \\ &+\mathrm{ND\_ti} \times \mathrm{et(t_i, p_m(v_{higest}, f_{higest}))} \end{aligned} \tag{14}$$

When there are no processing and execution tasks, processors switch to idle mode, where the energy consumed by processors is calculated by Equation 15. Where m is the total number of processors, makespan is the maximum time for completion of tasks by processors, also known as scheduling length.

$$\begin{aligned} &\mathrm{E_{processor.idle}} \\ &= ACv_{lowest}^2 f_{lowest} \Bigg(|m| \times makespan \\ &- \sum_{i=1}^{n} \left(\sum_{j=1}^{k} et(t_i, P_m(v_j, f_j)) \right) \\ &+ND\_ti \times et(t_i, p_m(v_{highest}, f_{highest})) \end{aligned} \tag{15}$$

Finally, the total energy consumed by processors to execute the task dependency graph can be obtained through sum of Equations 14 and 15.

$$\mathrm{E_{processors}} = \mathrm{E_{processors.active}} + \mathrm{E_{processors.idle}} \tag{16}$$

3.5.3.2. Communication energy

Since the processors in each datacenter have been assumed to be homogeneous, the data transfer speed and power consumption are identical.

The communication energy consumed by edge $d_{ij} \in D$ can be denoted as ec_{ij}, where PC is the power of interconnect. ec_{ij} is calculated through Equation 17.

$$\mathrm{ec}_{ij} = \mathrm{PC} \times \mathrm{ct}(\mathrm{d}_{\mathrm{ij}}) \tag{17}$$

Therefore, the total communication energy for the entire network can be calculated through Equation 18.

$$E_{Communications} = \sum_{i=1}^{n} \sum_{v_j \in Succ(v_i)} (x_{ij} \times ec_{ij}) \tag{18}$$

In Equation 18, element $x_i j$ is expressed by Equation 19 below:

$$x_{ij} = \begin{cases} 0 \;\; if\ (t_i^{end},\ p_m) = (t_j^{st},\ p_m) \\ 1 \;\; O.W \end{cases} \tag{19}$$

Finally, the total energy consumed by cloud datacenters can be obtained through sum of Equations 16 and 18.

$$\begin{aligned} E_{Total} \;&= E_{dynamic(processor.active)} \\ &\quad + E_{dynamic(processor.idel)} \\ &\quad + E_{communication} \end{aligned} \tag{20}$$

4. Proposed method

This section describes the proposed algorithm (EATSDCD) for scheduling dependent tasks, in order to mitigate energy consumption under throughput and makespan constraints. Based on the performance and energy models shown in Section 3, we can demonstrate the effects of combined duplication and clustering strategy together with the DVFS technique to achieve the stated objectives. The new algorithm consists of two phases namely EATSDC and EADVFSA.

In the first phase, a schedule serves to reduce communication energy and increase throughput. It is obtained through the energy-aware task duplication-clustering algorithm (EATDC). The second phase focuses on implementation of DVFS technique for each processor to decrease computation energy consumption of DAG, using the energy-aware dynamic voltage/frequency scaling algorithm (EADVFSA). The following sub-section describes both phases in greater details.

Algorithm 1. ***Energy − Aware Task Scheduling with Duplication − Clustering Algorithm (EATSDC)***

Input: Task Dependency Graph; DAG(T,D), Energy constraint; $\boldsymbol{En_c}$, Throughput constraint; $\boldsymbol{Th_c}$
Output: DAG′(T,D)

1. Begin
2. DAG′(T,D) ← DAG(T,D)
3. Task Clustering $C'_i \leftarrow \{C_i | C_i = \{t_i\}$ for all $t_i \in T\}$, C'_i is an unordered list of task-clusters, initially each task t_i a separate task-Cluster C_i
4. For all task-cluster C'_i, if $\boldsymbol{Th_c} > 0$ then number of$(C'_i) \leftarrow \boldsymbol{Th_c} \times$ et$(C_i,\ p_j)$ else number of$(C'_i) \leftarrow 1$
5. $\boldsymbol{Th_{cur}} \leftarrow$ calculate throughput(DAG'(T,D))
6. If $\boldsymbol{Th_{cur}} < \boldsymbol{Th_c}$ then
7. For all edge $\boldsymbol{d_{ij}}$ with min(numr($\boldsymbol{t_i}$, numr($\boldsymbol{t_j}$)/$\boldsymbol{ct(d_{ij})} < \boldsymbol{Th_c}$
8. D' ← {for all $\boldsymbol{d_{ij}} \in$ D & $\boldsymbol{t_i}$ and $\boldsymbol{t_j}$ belong separate cluster}
9. Sort the edges $\boldsymbol{d_{ij}}$ of the DAG in a descending order of edge time.
10. Initially all edges are unexamined.
11. Repeat:
12. Pick an unexamined edge which has largest edge time and CommR$(d'_{ij}) < \boldsymbol{Th_c}$, mark it as examined
13. t'_i is the source task and t'_j is destination task of d'_{ij}
14. DAG_1(T,D) ← DAG′(T,D), DAG_2(T, D) ← DAG′(T,D)
15. Zero the highest edge weight in DAG_1(T,D)
16. Duplicate (t'_i(DAG_2)) and zero the highest edge time DAG_2(T, D)
17. Constraint Critical Path(CP):
18. If ($CP_1 < CP_2$) then $DAG'(T, D) \leftarrow DAG_1(T, D)$ else $DAG'(T, D) \leftarrow DAG_2(T, D)$
19. $\boldsymbol{En_{cur}} \leftarrow$ calculate energy consumption $DAG'(T, D)$
20. end
21. While ($\boldsymbol{En_{cur}} > \boldsymbol{En_c}$) do
22. D′ ← {for all $\boldsymbol{d_{ij}} | \boldsymbol{d_{ij}} \in$ D & $\boldsymbol{t_i}$ and $\boldsymbol{t_j}$ belong separate cluster}
23. If D′ = Φ then return null;
24. List D′ ← sort the remaining edge of the DAG after Clustering & Duplication in descending order of edge communication time.
25. $d'_{ij} \leftarrow$ select the first edge in D′,
26. zeroing the d′ij
27. $\boldsymbol{En_{cur}} \leftarrow$ compute energy consumption DAG′(T,D)
28. End
29. When two cluster are merged, the ordering of tasks in the resulting cluster is based on their b-level (algorithm 2).
30. END

4.1. Energy-Aware Task Duplication-Clustering Algorithm (EATSDC)

The first phase presents the Energy-Aware Task Duplication-Clustering Algorithm (EATSDC) for parallel task scheduling. Our EATSDC attempts to satisfy makespan, throughput and energy constraints

using duplication and clustering strategy. Task clustering reduces makespan by zeroing edges of high communication time and proper adoption of the strategy. Task duplication decreases the communication overhead by reducing allocating certain tasks to multiple processors and thereby mitigate energy consumption. The pseudo-code of EATSDC is shown in Algorithm 1.

4.1.1. Generate original task scheduling sequence

Given that one of the scheduling objectives is to reduce makespan, task scheduling based on descending order of their b-level can lead to earlier scheduling of tasks on a critical path.

In fact, b-level is a priority assigned to each task. The b-level of task t_i is the length of longest path from t_i to an exit node. The b-level of a task is bounded from above by the length of a critical path. b-level is calculated through Algorithm 2.

Algorithm 2. Computation of b − level

```
1.  Begin
2.    Construct a list of all tasks ∈ T in reversed topological
      order, call it RevTopList.
3.    For each task t_i in RevTopList do
4.      max_length = 0
5.      For each immediate succeeding task t_j of task t_i do
6.        If ct(d_ij) + b − level(t_j) > max_length then
7.          max_length = ct(d_ij) + b − level(t_j)
8.        end if
9.      end for
10.   b − level{t_j} = et{t_i, p_j} + max_length
11. end for
12. end
```

4.2. Energy-Aware Dynamic Voltage/Frequency Scaling Algorithm (EADVFSA)

After applying the duplication and clustering strategy on the input graph, which leads to lower energy consumption and makespan, and also higher throughput, we intend to further mitigate the energy consumed by processors by determining the critical path and non-critical paths. We also specify the slack time of non-critical tasks, and calculate the voltage and frequency of processors assigned to processing of tasks in non-critical paths as well as idle and communication phases through scaling down DVFS. For this reason, it is essential to first explore the important parameters used in applying DVFS techniques to reduce energy consumption. These parameters have been listed in Table 3.

Table 3
Important parameters used in DVFS technique

Notations	Definition
$EST(t_i)$	*Earliest start time of task* t_i
$\boldsymbol{EFT(t_i)}$	*Earliest finish time of task* t_i
$LST(t_i)$	*Latest start time of task* t_i
$\boldsymbol{LFT(t_i)}$	*Latest finish time of task* t_i

4.2.1. Calculation of slack time for a task

The parameters in Table 3 are used to calculate the slack time of tasks and determine the critical path. Calculation of Earliest Start Time (EST) is a top-down method, which starts with the first task and ends with the last task, calculated by Equation 21.

$$\mathrm{EST}(t_i) = \begin{cases} 0 & \text{if } pred(t_i) = \phi \\ \mathrm{MAX}(EFT(t_j), & \\ \mathrm{MAX}(EFT(t_k) + (d_{ki})) & \text{O.W} \end{cases} \quad (21)$$

$$t_k \epsilon pred(t_i), d_{ii} \epsilon D$$

After calculating EST, the Earliest Finish Time (EFT) can be calculated for task t_i by Equation 22.

$$EFT(t_i) = EST(t_i) + et(t_i, p_m) \quad (22)$$

Moreover, the calculation of Last Finish Time (LFT) is a bottom-up method, which starts with the last task and ends with the first task, calculated by Equation 23.

$$\mathrm{LFT}(t_i) = \begin{cases} EFT(t_j) \text{ or makespan} & \text{if } \succ(t_i) = \phi \\ \mathrm{Min}(LST(t_j), & \\ \mathrm{Min}(LST(t_k) - (d_{ik})) & \text{O.W} \end{cases} \quad (23)$$

$$t_k \in succ(t_i), d_{ij} \in D$$

The Latest Start Time (LST) for task t_i is also calculated by Equation 24.

$$LST(t_i) = LFT(t_i) - et(t_i, p_m) \quad (24)$$

4.2.2. Determining the critical path

Critical path is the longest path through a DAG from entry task to exit task. It consists of the set of tasks that, if delayed in any way, would cause a delay in completion of the all tasks. The tasks, whose LST is equal to their EST, make up the critical path (or, equivalently, whose LFT is equal to their EFT).

4.2.3. Calculating the slack time of tasks

The non-critical tasks in a DAG are distinguished by the presence of slack. Slack is the amount of time by which the start of an activity can be delayed without delaying the makespan. Critical tasks have zero slack, while non-critical tasks have slack value, This is known as slack time. For each task ith, slack time is calculated through Equation 25.

$$\textit{Slack time for task } t_i = LST(t_i) - EST(t_i) \textit{ or } LFT(t_i) - EFT(t_i) \quad (25)$$

The pseudo-code of calculating slack time, critical and non-critical path have been described in Algorithm 3.

Algorithm 3. ***Calculate Slack time & Critical Path***

```
1.  Begin
2.  Initially all tasks in a descending order according to their
      finishing time (Queue topological sort)
3.  For each task t_i in DAG do
4.      Calculate EST(t_i), EFT(t_i), LST(t_i), LFT(t_i) as
      Equations (21–24)
5   end for
6.  For each task t_i in Queue topological sort do
7       Calculate Slack time of t_i as Equation (25)
8.      if Slack time task t_i = 0 then
9.          Add task t_i to Critical Path List
10.     else
11.         Add task t_i to Non-Critical Path List
12.     end if
13. end for
14. end
```

4.2.4. Voltage/frequency scaling

This section shows how to employ the DVFS technique to scale down the voltage/frequency of processors assigned to non-critical tasks, reduce the idle and communication phases, and scale up the voltage/frequency of processors assigned to critical tasks, thereby to mitigate energy consumption.

The critical path (CP) of scheduled task graph in a Gantt chart is a set of time slots of task execution and data communication from the first task to the last task, of which the sum of computation time and communication time is the makespan. Assuming that The CP is $t_1 - t_3 - t_5 - t_6$, the best-effort scheduling algorithm does not extend the makespan, the voltage/frequency of processors during the time slots of task execution in the CP is not changed. Voltage and frequency of other time slots in a Gantt chart are considered to be scaled down. $\text{Processor}'_k.\text{freq}^{\text{op}}$, is calculated as shown in Equation 26:

$$\text{Processor}'_k.\text{freq}^{\text{op}} = \text{freq}_{\text{highest}} \times \frac{et(t_i,\ p_m(v_{\text{highest}},\ f_{highest}))}{slak\ time\ for\ task\ t_i} \quad (26)$$

The pseudo-code of voltage/frequency scaling is shown in Algorithm 4.

Algorithm 4. ***voltage/frequency scaling***

```
1.  Begin
2.  for each proc_j do
3.   for each times slot in proc'_j s do
4.    if proc_j execute a critical task t_i then
5.     scale up proc_j frequency to highest
6.    end if
7.    if proc_j execute a non critical task t_i then
8.   calculate proc_j frequency to proc_j.f^op as Equation (26)
9.    end if
10.  if proc_j is idle or it executes a communication phase then
11.    scale down proc_j frequency to lowest
12.   end if
13.  end for
14.  end for
15.  end
```

5. Time complexity analysis

Given that the input of the proposed algorithms is DAG (T, D), in which |T| and |D| represent the number of tasks and edges, respectively, we want to analyze the time complexity of algorithms presented in the previous sections.

5.1. Analysis of EATSDC

5.1.1. Algorithm 1

This algorithm executes the clustering and duplication strategies on the input graph. Lines 3 and 4 require |T|, while Lines 5 to 8 require |D| operation times to calculate CompR(G) and CommR(G). Sorting in Line 9 can be done at time |D| log |D| based on *quick sort*. Lines 10 to 16 require $2 \times (|T| + |D|)^2$ operations, calculation of CP in Line 17 requires $(|T| + |D|)^2$ operations, and Lines 19 to 25 require $|D|^2$ operations to satisfy the energy consumption. Moreover, the calculation of b-level for all tasks in the integrated cluster requires $(|T| + |D| + |D|)$ operations. As a result, the time complexity of EATDCA is equal to $O\left(|D| \times (|T| + |D|)^2\right)$.

5.1.2. Algorithm 2

This algorithm computes the b-level for tasks. The construct RevTopList of Line 2 can be done in $o(|T| + |D|)$ time. Lines 3 and 5 are a double-loop.

Given that the number of iterations in the two loops is equal to the number of children of each node to which they are connected, the total duplications of Lines 3-11 is |D| times. Therefore, the total number of iterations of the algorithm is equal to $(|T| + |D| + |D|)$, with time complexity of $0(|T| + |D|)$.

5.2. Analysis of EADVFSA

5.2.1. Algorithm 3

This algorithm computes the slack time for task and then produces a CP. The sorting of Line 2 can be done in |T| log |T| time using quick sort. Assuming task has k successors or predecessors, Lines 3–5 occur k |T| times. Lines 6–12 compute the slack time for each task in the DAG, determine CP, and execute |T| times. Thus, the total number of iterations is equal to (|T| log |T| + k |T| + |T|) and the complexity for algorithm 3 is $O(|T| \log |T|)$.

5.2.2. Algorithm 4

This algorithm scales the frequency of processors. Assuming each virtual machine has |m| processors, with s time slots, Line 2 and 3 are double-loop. Hence, the complexity of Algorithm 4 is O (s |m|).

6. Performance analysis with simulation

This section presents the experiments carried out to evaluate the proposed heuristic algorithm, Energy-Aware Task Scheduling with Duplication Clustering Dynamic voltage/frequency Algorithm (EATSDCD) and compare EATSDCD against previous work, namely power aware task clustering (PATC) [20], power aware list-based scheduling (PALS) [20], Energy Aware Duplication Scheduling (EADUS) & TEBUS [44] with objective energy saving, RASD [33], Heterogeneous Earliest Finish Time (HEFT) [51] with objective execution time and PaPilo [41], Throughput Constrained Latency Optimization heuristic (TCLO) & Throughput Constrained Latency Optimization Pipelined (TCLO-P) [8] in homogeneous and heterogeneous environments while considering energy consumption and throughput.

6.1. Energy, throughput and makespan constraint settings

The solution described in the proposed method for best-effort scheduling optimizes energy consumption while meeting makespan and throughput requirements. We define lower bound of the energy constraint denoted as $\mathrm{En}_{\min}$ and upper bound of throughput constraint, denoted as $\mathrm{Th}_{\max}$ and lower bound of the makespan or critical path, denoted as $\mathrm{CP}_{\min}$.

The minimum consumed energy by processors to execute tasks occurs when the communication cost is excluded and no duplication takes place $(\mathrm{numr}(t_i) = 1)$. To that end, it is essential to execute all tasks on a single processor.

Therefore, the minimum energy consumption can be calculated as shown in Equation 27.

$$En_{\min}(G) = \sum_{\forall C_i \in C} et(C_i) \tag{27}$$

Moreover, the maximum throughput is achieved when we have |m| processors in the system as indicated by Equation 28.

$$Th_{\max}(G) = \frac{|m|}{\sum \forall C_i \in Cet(C_i)} \tag{28}$$

The minimum makespan or critical path, denoted as $\mathrm{CP}_{\min}$(G), is achieved by clustering all tasks on one processor.

This discard the communication time and achieves the makespan constraint, as represented in Equation 29.

$$CP_{\min}(\mathrm{G}) = \mathrm{EFT}(t_{exit}) \tag{29}$$

Since it is impossible to simultaneously obtain the minimum energy consumption and maximum throughput, it is crucial to consider a coefficient for $En_{\min}(G)$ and $Th_{\max}(G)$.

For that purpose, three energy constraints $En_{\min}(G)$ are set: $1.2En_{\min}(G)$, $1.5En_{\min}(G)$ and $2.0En_{\min}(G)$ and three throughput constraints $Th_{\max}(G)$.

$$0.25Th_{\max}(G), 0.5Th_{\max}(G) \textit{ and } 0.75Th_{\max}(G).$$

6.2. Randomly generated application task graphs

In this paper, we first considered the randomly generated application task graph. Currently, there are many random graph generator tools to generate

weighted application DAG, such as STG (standard task graph) [45]. STG is a kind of benchmark for evaluation of proposed scheduling algorithms. Three fundamental characteristics of the DAG are considered:

- DAG size, (n): The number of tasks in DAG.
- Communication-to-Computation Ratio, (CCR): it is the ratio between average communication time to the average computation time of the application DAG. Equation 30 describes how the CCR of a DAG is calculated.

$$CCR = \frac{\sum_{1 \le i,j \le n^{ct}(d_{ij})}}{\sum_{1 \le i \le n^{w}i}} \tag{30}$$

- Parallelism factor, (λ): the number of levels of the application DAG.

In our simulation experiments, the DAG sizes vary between 40 to 1000, in steps of 40, with random node and edge weights. CCR was varied as 0.1, 1 and 10. The number of levels is determined by λ, which varied as 0.5, 1.0, 2.0 and 5.0. in total about 200 graphs were generated for evaluation of the proposed method with other algorithms.

6.3. Experimental results

The platform of simulation environment to evaluate our work is CloudSim toolkit [46] based on Java, which supports the modeling and simulation of energy-aware computational resources in large-scale cloud-computing datacenters. We installed CloudSim in an Asus Notebook with Intel core i7-A540UP CPU 2.4 GHz with 8 cores and 4GB of memory. We create five datacenters in our simulation, and set 200 virtual machines, each involving three processor types namely AMD Turion 64 MT-34, AMD Opteron 2218 and Intel core i3-540 respectively [34]. These are all equipped with DVFS technology. Table 4 shows the details of the four processor types.

Firstly, we compare the proposed EATSDCD against the other four algorithms namely PALC, PATC, EADUS & TEBUS. According to the simulation results, the parameters of DAG size and CCR can greatly affect the extent of energy saving.

For CCR = 10 and CCR = 0.1, the application DAG is computation intensive and communication intensive respectively. The energy-saving of EATSDCD is higher than that of other algorithms. As for CCR = 1, the energy-saving of EATSDCD and other four algorithms are almost equal. Table 5 compares EATSDCD against other energy-aware DAG scheduling algorithms in term of max energy saving. PALC and PATC use the clustering and DVFS technique for parallel task scheduling in cluster to reduce energy consumption. EADUS and TEBUS use the duplication and clustering technique for scheduling precedence-constrained parallel tasks on clusters to balance scheduling length and energy consumption.

The second set of simulation is to compare the proposed EATSDC algorithm against the other three algorithms namely RASD [33], HEFT [51]. According to the simulation results, Figs. 5–7 show the makespan of the EATSDC varies with respect to the DAG size (40, 80, 120, 160, 200) and the CCR size (0.1, 1, 5).

The third set of simulation shows the performance of the EATSDCD algorithm with CCR = 1 and throughput set to $0.25Th_{\max}(G)$, $0.5Th_{\max}(G)$ and $0.75Th_{\max}(G)$. against previously proposed schemes: PaPilo [41], TCLO & TCLO-P [8].

Table 4
Power consumption for different voltage/frequency of processors [34]

Processors	AMD Opteron 2218	AMD Turion MT-34	Intel Core i3-540
Voltage (V)	1.1, 1.15, 1.15, 1.20, 1.25, 1.30	0.9, 1.0, 1.05, 1.1, 1.15, 1.2	1.125, 1.125, 1.2, 1.2, 1.3, 1.3, 1.375
Frequency (GHz)	1.0, 1.8, 2.0, 2.2, 2.4, 2.6	0.8, 1.0, 1.2, 1.4, 1.6, 1.8	3.07, 3.2, 3.4, 3.6, 3.8, 4.0, 4.2
Highest power (W)	95	25	108
Lowest power (W)	26.16	6.25	53

Table 5
Comparison of energy-saving between our proposed EATSDCD and the other four algorithms

Energy-aware DAG scheduling algorithms	Technique	Maximum energy saving (%)
PATC [20]	DVFS & Clustering	39.7
PALS [20]	DVFS & ETF scheduling	44.3
EADUS & TEBUS [44]	Clustering & Duplication	16.8
EATSDCD (proposed method)	Duplication & Clustering & DVFS	52.7

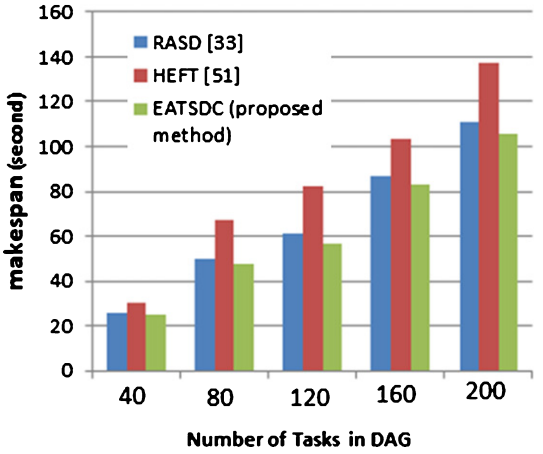


Fig. 5. Comparison of makespan between EATSDC and RASD, HEFT for CCR = 0.1.

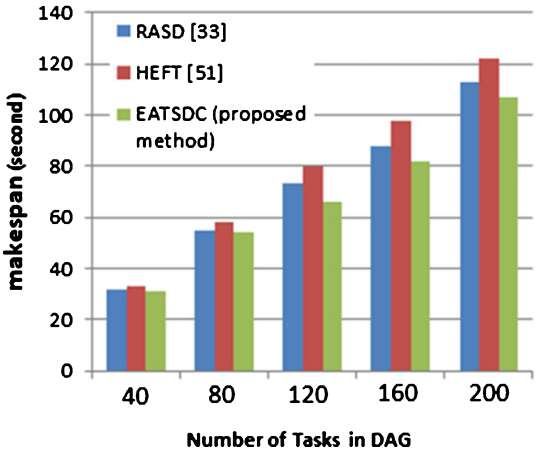


Fig. 6. Comparison of makespan between EATSDC and RASD, HEFT for CCR = 1.

The results are shown in Table 6. Based on the simulation observations, we can find that the PaPilo algorithm and the proposed EATSDCD algorithm achieved the $En_{cur} = 1.5\ En_{min}(G)$ and $En_{cur} = 2.0\ En_{min}(G)$ energy and throughput constraints for all 200 graph samples.

Meanwhile, as the energy constraints changed to $En_{cur} = 1.2\ En_{min}(G)$ at the same throughput constraints, only EATSDCD achieved the specified constraint in all 200 sample graphs. The objectives were realized because after applying the duplication and clustering technique on the input graph samples and calculation of slack time for all tasks, the DVFS technique was adopted to mitigate the voltage and frequency of processors assigned to process non-critical tasks as well as idle and communication phases. This in turn minimized the energy consumption and thus fulfilled the specified constraints.

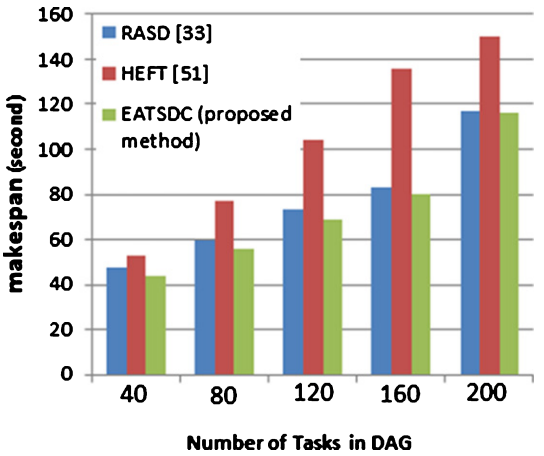


Fig. 7. Comparison of makespan between EATSDC and RASD, HEFT for CCR = 5.

Table 6
Number of feasible solutions for 100-node random task graph, with different Energy and throughput constraints, for CCR = 1

	(*a*) $En_{cur} = 2.0\ En_{min}(G)$			
Th_{max}	EATSDCD(proposed method)	TCLO [8]	TCLO-P [8]	PaPilo [41]
$0.25Th_{max}$	1	0.96	0.72	1
$0.5Th_{max}$	1	0.82	0.64	1
$0.75Th_{max}$	1	0.77	0.57	1
	(*b*) $En_{cur} = 1.5\ En_{min}(G)$			
Th_{max}	EATSDCD(proposed method)	TCLO [8]	TCLO-P [8]	PaPilo [41]
$0.25Th_{max}$	1	0.51	0.69	1
$0.5Th_{max}$	1	0.47	0.58	1
$0.75Th_{max}$	1	0.39	0.55	1
	(c) $En_{cur} = 1.2En_{min}(G)$			
Th_{max}	EATSDCD(proposed method)	TCLO [8]	TCLO-P [8]	PaPilo [41]
$0.25Th_{max}$	1	0.43	0.51	0.91
$0.5Th_{max}$	1	0.37	0.44	0.83
$0.75Th_{max}$	1	0.23	0.36	0.78

The results of simulation indicated that the newly proposed algorithm can save greater energy than other algorithms for the following reasons.

- EATSDCD uses task duplication and clustering to reduce the necessary communication between processors.
- EATSDCD reduce the energy consumption during the communication phase.
- EATSDCD reduce energy consumption when processors are idle.
- EATSDCD employs the DVFS technique to extend the task slack time.

7. Conclusions and future work

In this paper, we proposed a novel green energy-aware scheduling algorithm, called Energy Aware Task Duplication Clustering Dynamic voltage/frequency scaling (EATSDCD). It employs the clustering & duplication technique on homogeneous/heterogeneous DVFS-enabled cloud datacenter processors. EATSDCD can be applied to application DAGs such as STG so as to optimize energy efficiency at the premise of meeting the throughput and makespan constraints. In the first phase, a schedule serves to reduce communication energy and increase throughput. It is obtained through the energy-aware task duplication-clustering algorithm (EATDCA). The second phase focuses on implement ion of DVFS technique for each processor that can scale down clock frequency and supply voltage whenever tasks have slack time and during idle and communication time slots to decrease energy consumption of DAG, using the Energy-Aware Dynamic voltage/frequency scaling Algorithm (EADVFSA).

testbed is developed and different parameters are tested on the randomly generated DAG to evaluate and illustrate the superiority and effectiveness of EATSDCD. It was also compared against duplication and clustering-based algorithms and DVFS-based algorithms. In terms of energy consumption and makespan, the results show that our proposed algorithm can save up to 8.3% and 20% energy compared against PALS [20] and PATC [20] algorithms without performance loss, respectively. Furthermore, there is 16% improvement over PaPilo [41] algorithm with $\mathrm{En}_{\mathrm{cur}} = 1.2\mathrm{En}_{\mathrm{min}}(\mathrm{G})$. In comparison with RASD [33] and HEFT [51] algorithm, the execution time has been reduced in heterogeneous environments.

The future works can be divided into three areas. Firstly, the user and service provider initiate negotiations to reach a green SLA concerning the makespan extension rate. An agreement on η rate (makespan $\leq (1+\eta)\times$ makespan best) will achieve the service quality parameters and minimize energy saving by up to 52.7% in the newly proposed method. Secondly, a few samples of real applications, such as MPEG-2 decoder [41], can be run through the new algorithm. Thirdly, a model can be proposed to mitigate energy consumption in task scheduling for other components such as disk, memory and network.

Acknowledgments

The authors would like to thank the anonymous reviewers and the editor for their insightful comments and suggestions.

References

[1] M. Uddin, Y. Darabidarabkhani, A. Shah and J. Memon, Evaluating power efficient algorithms for efficiency and carbon emissions in cloud data centers: A review, *Renewable and Sustainable Energy Reviews* **51** (2015), 1553–1563.

[2] A. Dobhal, Improved real-time energy aware parallel task scheduling in a cluster, In: *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on*, 2016, pp. 475–480. IEEE.

[3] M. Mezmaz, N. Melab, Y. Kessaci, Y.C. Lee, E.-G. Talbi and A.Y. Zomaya and D. Tuyttens, A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems, *Journal of Parallel and Distributed Computing* **71**(11) (2011), 1497–1508.

[4] S. Smanchat and K. Viriyapant, Taxonomies of workflow scheduling problem and techniques in the cloud, *Future Generation Computer Systems* **52** (2015), 1–12.

[5] J.D. Ullman, NP-complete scheduling problems, *Journal of Computer and System Sciences* **10**(3) (1975), 384–393.

[6] O. Sinnen, A. To and M. Kaur, Contention-aware scheduling with task duplication, *Journal of Parallel and Distributed Computing* **71**(1) (2011), 77–86.

[7] S.C. Kim, S. Lee and J. Hahm, Push-pull: Deterministic search-based dag scheduling for heterogeneous cluster systems, *IEEE Transactions on Parallel and Distributed Systems* **18**(11) (2007).

[8] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Sadayappan and J. Saltz, Optimizing latency and throughput of application workflows on clusters, *Parallel Computing* **37**(10) (2011), 694–712.

[9] B. Vázquez-Barreiros, M. Mucientes and M. Lama, Enhancing discovered processes with duplicate tasks, *Information Sciences* **373** (2016), 369–387.

[10] D. Boru, D. Kliazovich, F. Granelli, P. Bouvry and A.Y. Zomaya, Energy-efficient data replication in cloud computing datacenters, *Cluster Computing* **18**(1) (2015), 385–402.

[11] F.A. Omara and M.M. Arafa, Genetic algorithms for task scheduling problem, *Journal of Parallel and Distributed Computing* **70**(1) (2010), 13–22.

[12] J. Mei, K. Li and K. Li, A resource-aware scheduling algorithm with reduced task duplication on heterogeneous computing systems, *The Journal of Supercomputing* **68**(3) (2014), 1347–1377.

[13] K. Shin, M. Cha, M. Jang, J. Jung, W. Yoon and S. Choi, Task scheduling algorithm using minimized duplications in homogeneous systems, *Journal of Parallel and Distributed Computing* **68**(8) (2008), 1146–1156.

[14] Y. Ding, X. Qin, L. Liu and T. Wang, Energy efficient scheduling of virtual machines in cloud with deadline constraint, *Future Generation Computer Systems* **50** (2015), 62–74.

[15] W. Chen, R.F. da Silva, E. Deelman and R. Sakellariou, Using imbalance metrics to optimize task clustering in scientific workflow executions, *Future Generation Computer Systems* **46** (2015), 69–84.

[16] F. Zhang, J. Cao, K. Li, S.U. Khan and K. Hwang, Multi-objective scheduling of many tasks in cloud platforms, *Future Generation Computer Systems* **37** (2014), 309–320.

[17] E. Arianyan, H. Taheri and V. Khoshdel, Novel fuzzy multi objective DVFS-aware consolidation heuristics for energy and SLA efficient resource management in cloud data centers, *Journal of Network and Computer Applications* **78** (2017), 43–61.

[18] N.B. Rizvandi, A.Y. Zomaya, Y.C. Lee, A.J. Boloori and J. Taheri, Multiple frequency selection in DVFS-enabled processors to minimize energy consumption, *arXiv preprint arXiv:1203.5160*.

[19] Y. Hu, C. Liu, K. Li, X. Chen and K. Li, Slack allocation algorithm for energy minimization in cluster systems, *Future Generation Computer Systems* **74** (2017), 119–131.

[20] L. Wang, K. Su, D. Chen, J. Kołodziej, R. Ranjan, C.-Z. Xu and A. Zomaya, Energy-aware parallel task scheduling in a cluster, *Future Generation Computer Systems* **29**(7) (2013), 1661–1670.

[21] R. Entezari-Maleki, L. Sousa and A. Movaghar, Performance and power modeling and evaluation of virtualized servers in IaaS clouds, *Information Sciences* **394** (2017), 106–122.

[22] J. Mei, K. Li and K. Li, Energy-aware task scheduling in heterogeneous computing environments, *Cluster Computing* **17**(2) (2014), 537–550.

[23] Y.-C. Ouyang, Y.-J. Chiang, C.-H. Hsu and G. Yi, An optimal control policy to realize green cloud systems with SLA-awareness, *The Journal of Supercomputing* **69**(3) (2014), 1284–1310.

[24] C.-H. Lin and J.-C. Ke, Multi-server system with single working vacation, *Applied Mathematical Modelling* **33**(7) (2009), 2967–2977.

[25] M. Jain and A. Jain, Working vacations queueing model with multiple types of server breakdowns, *Applied Mathematical Modelling* **34**(1) (2010), 1–13.

[26] F. Juarez, J. Ejarque and R.M. Badia, Dynamic energy-aware scheduling for parallel task-based application in cloud computing, *Future Generation Computer Systems* (2016).

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

[28] A. Beloglazov, J. Abawajy and R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, *Future Generation Computer Systems* **28**(5) (2012), 755–768.

[29] N.B. Rizvandi, J. Taheri and A.Y. Zomaya, Some observations on optimal frequency selection in DVFS-based energy consumption minimization, *Journal of Parallel and Distributed Computing* **71**(8) (2011), 1154–1164.

[30] S. Baskiyar and R. Abdel-Kader, Energy aware DAG scheduling on heterogeneous systems, *Cluster Computing* **13**(4) (2010), 373–383.

[31] H. Kimura, M. Sato, Y. Hotta, T. Boku and D. Takahashi, Emprical study on reducing energy of parallel programs using slack reclamation by dvfs in a power-scalable high performance cluster, In: *Cluster Computing, 2006 IEEE International Conference on*, IEEE, 2006, pp. 1–10.

[32] V.K. Mohan Raj and R. Shriram, Power management in virtualized datacenter – A survey, *Journal of Network and Computer Applications* **69** (2016), 117–133.

[33] X. Tang, K. Li, R. Li and B. Veeravalli, Reliability-aware scheduling strategy for heterogeneous distributed computing systems, *Journal of Parallel and Distributed Computing* **70**(9) (2010), 941–952.

[34] W. Liu, W. Du, J. Chen, W. Wang and G. Zeng, Adaptive energy-efficient scheduling algorithm for parallel tasks on homogeneous clusters, *Journal of Network and Computer Applications* **41** (2014), 101–113.

[35] M. Hu, J. Luo, Y. Wang and B. Veeravalli, Adaptive Scheduling of Task Graphs with Dynamic Resilience, *IEEE Transactions on Computers* **66**(1) (2017), 17–23.

[36] I. Ahmad and Y.-K. Kwok, On exploiting task duplication in parallel program scheduling, *IEEE Transactions on Parallel and Distributed Systems* **9**(9) (1998), 872–892.

[37] C.H. Papadimitriou and M. Yannakakis, Towards an architecture-independent analysis of parallel algorithms, *SIAM Journal on Computing* **19**(2) (1990), 322–328.

[38] J.-Y. Colin and P. Chrétienne, CPM scheduling with small communication delays and task duplication, *Operations Research* **39**(4) (1991), 680–684.

[39] Y.-C. Chung and S. Ranka, Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors, In: *Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, IEEE Computer Society Press, 1992, pp. 512–521.

[40] B. Kruatrachue and T. Lewis, Grain size determination for parallel processing, *IEEE Software* **5**(1) (1988), 23–32.

[41] C.-S. Lin, C.-S. Lin, Y.-S. Lin, P.-A. Hsiung and C. Shih, Multi-objective exploitation of pipeline parallelism using clustering, replication and duplication in embedded multi-core systems, *Journal of Systems Architecture* **59**(10) (2013), 1083–1094.

[42] R.W. Ahmad, A. Gani, S.H.Ab. Hamid, M. Shiraz, A. Yousafzai and F. Xia, A survey on virtual machine migration and server consolidation frameworks for cloud data centers, *Journal of Network and Computer Applications* **52** (2015), 11–25.

[43] A. Liang and Y. Pang, A novel, energy-aware task duplication-based scheduling algorithm of parallel tasks on clusters, *Mathematical and Computational Applications* **22**(1) (2016), 2.

[44] Z. Zong, A. Manzanares, B. Stinar and X. Qin, Energy-aware duplication strategies for scheduling precedence-constrained parallel tasks on clusters, In: *Cluster Computing, 2006 IEEE International Conference on*, IEEE, 2006, pp. 1–8.

[45] Standard Task Graph Set. <http://www.kasahara.elec.waseda.ac.jp/schedule/>.

[46] R. Buyya, R. Ranjan and R.N. Calheiros, Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities, In: *High*

Performance Computing & Simulation, 2009 HPCS'09 International Conference on, 2009, pp. 1–11. IEEE.

[47] A. Mohsenzadeh and H. Motameni, A trust model between cloud entities using fuzzy mathematics, *Journal of Intelligent & Fuzzy Systems* **29**(5) (2015), 1795–1803.

[48] H. Lei, R. Wang, T. Zhang, Y. Liu and Y. Zha, A multi-objective co-evolutionary algorithm for energy-efficient scheduling on a green data center, *Computers & Operations Research* **75** (2016), 103–117.

[49] S. Mustafa, B. Nazir, A. Hayat, A. ur Rehman Khan and S.A. Madani, Resource management in cloud computing: Taxonomy prospects, and challenges, *Computers and Electrical Engineering* **47** (2015), 186–203.

[50] A. Gerasoulis and T. Yang, A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors, *Journal of Parallel and Distributed Computing* **16** (1992), 276–291.

[51] H. Topcuoglu, S. Hariri and M.-Y. Wu, Performance-effective and low complexity task scheduling for heterogeneous computing, *IEEE Trans Parallel Distrib Syst* **13**(3) (2002), 260–274.

[52] Y. Sharma, B. Javadi, W. Si and D. Sun, Reliability and energy efficiency in cloud computing systems: Survey and taxonomy, *Journal of Network and Computer Applications* **74** (2016), 66–85.

[53] Y.C. Lee and A.Y. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, *IEEE Transactions on Parallel and Distributed Systems* **22**(8) (2011), 1374–1381.

[54] D. Bozdag, F. Ozguner and U.V. Catalyurek, Compaction of schedules and a two-stage approach for duplication-based DAG scheduling, *IEEE Transactions on Parallel and Distributed Systems* **20**(6) (2009), 857–871.

[55] W. Shu, W. Wang and Y. Wang, A novel energy-efficient resource allocation algorithm based on immune clonal optimization for green cloud computing, *EURASIP Journal on Wireless Communications and Networking* **2014**(1) (2014), 6.