Information Sciences 319 (2015) 113–131

Contents lists available at ScienceDirect

Information Sciences

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

Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster

Longxin Zhang [a], Kenli Li [a,*], Yuming Xu [a], Jing Mei [a], Fan Zhang [b,c], Keqin Li [a,d]

[a] College of Information Science and Engineering, National Supercomputing Center in Changsha, Hunan University, Changsha 410082, China
[b] Kavli Institute for Astrophysics and Space Research, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
[c] Research Institute of Information Technology, Tsinghua University, Beijing 100084, China
[d] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

A R T I C L E I N F O

Article history:
Received 29 June 2014
Received in revised form 7 February 2015
Accepted 14 February 2015
Available online 21 February 2015

Keywords:
Cluster computing
Energy consumption
List scheduling
Parallel application
Reliability

A B S T R A C T

A heterogeneous computing system in a cluster is a promising computing platform, which attracts a large number of researchers due to its high performance potential. High system reliability and low power consumption are two primary objectives for a data center. Dynamic voltage scaling (DVS) has been proved to be the most efficient technique and is exploited widely to realize a low power system. Unfortunately, transient fault is inevitable during the execution of an application while applying the DVS technique. Most existing scheduling algorithms for precedence constrained tasks in a multiprocessor computer system do not adequately consider task reliability. In this paper, we devise a novel Reliability Maximization with Energy Constraint (RMEC) algorithm, which incorporates three important phases, including task priority establishment, frequency selection, and processor assignment. The RMEC algorithm can effectively balance the tradeoff between high reliability and energy consumption. Our rigorous performance evaluation study, based on both randomly generated task graphs and the graphs of some real-world applications, shows that our scheduling algorithm surpasses the existing algorithms in terms of system reliability enhancement and energy consumption saving.

1. Introduction

Nowadays, big data applications are progressively becoming the major focus of attention due to the enormous increment of data generation and storage that has taken place in the recent years. The data size is constantly increasing, as of 2012 ranging from a few dozen terabytes to many petabytes of data in a single data set [37]. The major features of big data are high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery, and process optimization [5]. Many real-world areas such as telecommunications, health care, pharmaceutical, Internet search, financial and business informatics generate massive amounts of data. Tmall [38], which is the largest online shopping site in Asia, has become an indispensable part of daily life of Chinese. During the Chinese "Single Day" Double 11 shopping carnival, Alibaba made another record on November 11 2013: total transaction in a day hit 35 billion yuan (USD 5.71 billion). Merely 55 s after 0:00 on November 11, the transaction amount reached 100 million yuan (USD 16.3 million). At 0:06:07, it reached 1 billion yuan (USD 163 million). At 5:49 am, it reached 10 billion

* Corresponding author. Tel.: +86 731 88664161.
E-mail address: lkl@hnu.edu.cn (K. Li).

http://dx.doi.org/10.1016/j.ins.2015.02.023
0020-0255/© 2015 Elsevier Inc. All rights reserved.

yuan (USD 1.63 billion). About 188 million of transactions were conducted on this day. It contributes to the powerful support provided by Aliyun. In the data center of Aliyun, there are about 40 thousand jobs to be run on a cluster which is composed of 3000 nodes to process the 1.5 petabyte transaction records everyday, and then to output 20 terabyte results.

A data center garnered significant support and encouragement by its participants, who spanned industry, government labs, and academia [11]. To meet the needs of economic development, national centers for supercomputing have sprung up. The ranking of the top supercomputers in the world show a major trend in heterogeneous architectures. They are superior by power efficiency rather than speed. Such high end computing facilities can consume a very large amount of power, although they provide high performance computing solutions for scientific and engineering applications. For instance, operating a middle-sized data center (i.e., a university data center) demands 80,000 kW power [34]. In addition, high power consumption usually leads to expensive cooling costs. Furthermore, keeping computing facilities running on high power for a long time will result in high temperature of computing systems, which further degrades systems' availability and reliability.

While performance/hardware-cost has increased dramatically with the advancement of electronic technology, power consumption in computer systems has also increased according to Moore's law [36]. Such increased energy consumption causes severe ecological, economic, and technical issues. Hence, it is not very hard to image the size of adverse environmental footprint left by the heterogeneous computing systems (HCS) in a cluster. This issue has attracted extensive research activities in recent years, with the growing advocacy of green computing systems. Some hardware technologies [33], such as energy-efficient monitors, low-power microprocessors and processors consisting of multiple processing element cores and a selective-associative cache memory, are employed to address the energy consumption problems. Comprehensive surveys can be found in Refs. [33,2,4].

Task scheduling problems are classic and important. A large number of excellent algorithms are proposed. Huang et al. [15] proposed three types of fuzzy models to solve the Fuzzy Time-dependent Project Scheduling Problem (FTPSP) while guaranteeing resource constraints. Under the dynamic grid environment, Kołodziej and Khan [22] introduced a Hierarchic Genetic Strategy based Scheduler (HGS-Sched) to achieve fast reductions in makespan and flowtime in the concurrent search of the optimization domain. An adaptive scoring method was used to schedule jobs in grid environment by Chang et al. [7]. With the rapid development of society, reducing processor energy consumption has been a critically important and pressing research issue in recent years. The dynamic voltage and frequency scaling (DVFS) technique [35] is widely recognized as the basis of numerous energy management solutions. It exploits the fact that the dynamic power consumption is a strictly convex function of the CPU frequency, and attempts to conserve energy by reducing clock speed and supply voltage at active state. Benefiting from DVFS, various energy-aware task scheduling and resource management methods have emerged as promising studies for sustainable computing. Many excellent strategies and approaches have been developed, but their scope is restricted to unique processor systems [42], homogeneous computing systems [21,13,43], and battery based embedded systems [31].

Numerous algorithms have been devised to accomplish speedup for parallel applications in the form of directed acyclic graphs (DAG). It is generally appreciated that a task scheduling problem is NP-hard [12]. Usually, scheduling algorithms aim to map tasks onto proper processors and sort tasks in an appropriate sequence, so that task precedence constraints are met and the minimum scheduling length can be achieved. A significant number of existing studies are devised for homogeneous systems, such as the well known Dynamic-Level Scheduling (DLS) algorithm [27]. Recently, a few diverse list scheduling algorithms have been developed to handle heterogeneous systems, for instance, Constrained Earliest Finish Time (CEFT) algorithm [20], Critical-Path-On-a-Processor (CPOP) algorithm [32], and Heterogeneous Earliest Finish Time (HEFT) algorithm [32]. Among these algorithms, HEFT and CPOP have been proven to be very promising algorithms with their demonstrated low complexity and performance effective capability. It is widely accepted that a major challenge in scheduling is to diminish interprocessor communication cost. Node duplication is an effective solution that has been exploited to deal with the above described problem. Based on this, recently reported algorithms, such as HCPFD [14] and HLD [3], were proposed for HCS. They improve the performance by taking account of limited effective duplication into them. Idle time slots, scattered among the processors, are exploited for duplicating the immediate parent of a child task so as to make its start time earlier. The comparison analysis of HCPFD [14] and HLD [3] shows that they significantly outperform other algorithms, for instances, DLS [27] and HEFT [32]. However, the duplication technique aims to reduce the schedule length at the expense of sacrificing more energy and higher complexity. With respect to the promotion of system reliability, a number of hardware and software based techniques for hypercube (HC) fault tolerance was developed by Abd-El-Barr and Gebali [1], which can be exploited to enhance the system reliability and fault tolerance aspects of existing hypercube multi-computer networks (HCNs). Dogan and Özgüner developed three reliability cost functions that were incorporated into making dynamic level (DL) and introduced a Reliable Dynamic Level Scheduling (RDLS) algorithm [9,10]. Tang et al. proposed a Hierarchical Reliability-Driven Scheduling (HRDS) algorithm in a grid computing system [30]. Wang et al. developed scheduling heuristics which reduce energy consumption of parallel task execution in a cluster by using the DVFS mechanism [34].

Unfortunately, most of these approaches are on the basis of simple system models, which do not precisely reflect the real parallel computation systems. One of the assumptions, i.e., a node never fails during execution, may lead to some problems. In real systems, the transition faults in task execution are inevitable and may have an adverse impact on the running applications. Studies in [41] show that it is critical to design an accurate schedule with consideration of task reliability. Low power consumption and high system reliability, availability, and utility are the main concerns of modern high-performance computing system development. A number of studies [18,40,25] revealed the interplay between energy consumption and system reliability. However, these approaches are exclusively confined to the embedded systems. Li and Xu et al. analyzed the performance of heuristic power allocation and scheduling algorithms for precedence constrained parallel tasks [24,39]. Lee and Zomaya developed two energy-conscious scheduling algorithms which effectively balance the quality of schedules and energy consumption using dynamic voltage scaling (DVS) [23]. Tang et al. designed a reliability-driven scheduling architecture and proposed a reliability-aware scheduling algorithm for precedence constrained tasks [29]. Notwithstanding, none of them incorporates energy consumption and reliability together. In most cases, the scheduling length is not always as small as possible. It is crucial to run an application with higher reliability and lower energy consumption.

In this paper, we treat maximizing reliability with energy conservation for precedence constrained tasks in a heterogeneous cluster with dynamically variable voltage and speed as an combinatorial optimization problem. Our scheduling problem comprises three nontrivial subproblems, namely, precedence constraining, energy conserving, and reliability maximizing.

- *Precedence Constraining*. Compared to an independent task set, parallel tasks with precedence constraints make devise and analysis of heuristic scheduling algorithms particularly complicated.
- *Energy Conserving*. Processors should provide appropriate powers and energy efficient execution speeds, such that the schedule length is modest and the energy consumption is minimal.
- *Reliability Maximizing*. Tasks should be run at relatively high speeds without exceeding the maximum frequency of processors, such that the system reliability can be achieved optimal.

The above subproblems should be solved efficiently so that heuristic algorithms with overall fine performance can be explored. By adopting the DVS technique, three algorithms are presented in this paper, i.e., the Reliability-aware Heterogeneous Earliest Finish Time (RHEFT) algorithm, the Reliability-aware Critical-Path-On-a-Processor (RCPOP) algorithm, and a novel Reliability Maximization with Energy Constraint (RMEC) algorithm. Algorithms RHEFT and RCPOP are two intuitive strategies for system reliability enhancement. The main purpose is to lead to the presentation of algorithm RMEC. Each of these three algorithms comprises the following three phases. The task priority establishment phase builds a proper topological order for the application tasks. The frequency selection phase chooses an energy efficient frequency to execute each task. The processor assignment phase allocates a candidate task to a suitable processor, so as to get higher total system reliability with lower total energy consumption.

The rest of the paper is organized as follows. Section 2 presents the system, energy, application, and reliability models used in this paper. Based on the previously introduced reliability and energy models, Section 3 develops the RMEC algorithm and other two revised algorithms RHEFT and RCPOP. A simple motivational example is presented in Section 4. Extensive experimental results are discussed in Section 5. Finally, Section 6 gives concluding remarks and mentions future works.

2. Models

In this section, we introduce the system, power, application, fault and reliability models used in this paper.

2.1. System model

The system model used in this paper comprises a set PE of p heterogeneous cores/processors in a cluster. Each of them is available for DVFS technology; namely, each core can run at different speeds (i.e., different supply voltage levels). Each processor pe which belongs to set PE has f different available frequency levels (AFLs). It follows a random and uniform distribution among the four different sets of operation voltages/frequency. As clock frequency switching overhead takes an inappreciable amount of time (e.g., 10–150 μs [16,26]), such overhead is neglected in our paper. Besides, the communications among the processors are supposed to perform at the same speed on all links without contention. Our paper is based on the premise that the target system consists of a set of fully connected processors, which implies that each processor has a direct communication link to every other processor. Inter-processor communication is performed by a dedicated communication subsystem in such a way that is completely free of contention.

2.2. Power model

Generally, with the DVFS technique, the clock frequency is reduced alongside with the supply voltage for the approximately linear relationship between the supply voltage and operation frequency [6]. For the promising capability of energy saving, the DVFS technique is adopted in our study. It enables a processor dynamically adjust available frequency levels. To present a system-level power model, we adopt the classic one proposed in [44]. And the system power consumption is given as follows [41]:

$$P = P_s + \hbar(P_{ind} + P_d) = P_s + \hbar\left(P_{ind} + C_{eff}f^{\alpha}\right), \tag{1}$$

where P_s is the static power consumption, P_{ind} refers to the frequency-independent active power, and P_d represents the frequency-dependent dynamic power. The static power term, including the power to maintain the basic circuits, keeps the clock working and the memory staying in sleep mode, can be removed only by turning off the whole system. P_{ind} is a constant, independent of system operation frequency (i.e., the power consumption occurs while accessing external devices like main memory, I/O, and so on), can be decreased to a very small value by setting the system to standby mode [6]. P_d is the dynamic power dissipation, the dominant component of energy consumption in widely popular CMOS technology. It can be given by $P_d = C_{eff} \cdot V_{dd}^2 \cdot f$, where C_{eff} is the effective loading capacitance, V_{dd} is the supply voltage, and f is the clock frequency. Since $f \propto v^{\gamma}$ $(0 < \gamma < 1)$ [24], in other words, $v \propto f^{1/\gamma}$, we reckon that the frequency dependent active power consumption is $P_d \propto f^{\alpha}$, where $\alpha = 1 + 2/\gamma \geqslant 3$. In our studies, we have $P_d = C_{eff} f^{\alpha}$. And $\hbar$ indicates the system mode and represents whether active power consumption is occurred present. Particularly, $\hbar = 1$ signifies that the system is active currently. Otherwise, $\hbar = 0$ refers to a sleep mode that the system is in. In the context of this paper, all frequencies are normalized with respect to the maximum frequency $f_{\max}$ (i.e., $f_{\max} = 1.0$). And the energy consumption of task v_i can be calculated according to Eq. (2),

$$E_i(f_i) = P_{ind_i} \cdot \frac{c_i}{f_i} + C_{eff} \cdot c_i \cdot f_i^2, \tag{2}$$

where c_i is the computational cost at executing frequency of f_i. The total energy E_{total} consumed by processors during the execution of tasks in a task set is hence estimated by

$$E_{total} = \sum_{i=1}^{n} E_i(f_i). \tag{3}$$

For simplicity, only processor energy consumption is considered in this paper.

2.3. Application model

Generally, a parallel application program consisting of precedence constrained tasks can be represent by a directed acyclic graph (DAG). A DAG, $G = \langle T, E \rangle$, where T is the task set which comprises $|T|$ tasks that can be executed on any available processors. Set E is composed of the edges which represent task precedence constrains. An edge $w_{i,j} \in E$ between task node i and node j, both of which perform on different processors, denotes the intertask communication.

For each node τ_i in a given task graph, the direct predecessors of which are denoted by $parent(\tau_i)$, which is a set $parent(\tau_i) = \{\forall \tau_p \in T | e_{p,i} \in E\}$. And its direct successors are denoted as $child(\tau_i)$. If a task has no any predecessor, namely, $parent(\tau_i) = \emptyset$, it is termed as an entry task. Likewise, if a task has no any successor, namely, $child(\tau_i) = \emptyset$, it is called an exit task. Without loss of generality, we assume that a DAG in our study has (or can be transformed to have) exactly one entry task τ_{entry} and one exit task τ_{exit}. The notations about the system and application models used in this paper are summarized in Table 1.

The weight on task τ_i is denoted as w_i, which represents the computation cost. In addition, the execution time of task τ_i on processor pe_j refers to $w_{i,j}$ and its average computation cost is denoted by $\bar{w}_i$. Similarly, the weight $c_{i,j}$ assigned to an edge represents the communication cost between two tasks τ_i and τ_j. However, the communication occurs only when the two nodes are scheduled to two distinct processors. In other word, there is no communication cost provided that the two nodes are assigned to the same processor.

Consider the graph with eleven nodes as shown in Fig. 1, the edges, which are labeled with weights, reflect the communication costs of corresponding nodes in different processors. In our study, the target system comprises of a set PE of p

Table 1
Definitions of notations.

Notation	Definition
PE	A set of processing elements
V	A set of supply voltages
F	A set of supply frequencies
$w_{i,j}$	The computational cost of task $\tau_i \in T$ on processor $pe_j \in PE$
$c_{i,j}$	The communication cost between node τ_i and node τ_j
$\bar{w}_i$	The average computational time of a task when executed on different processors
$child(\tau_i)$	The set of immediate successors of task τ_i
$parent(\tau_i)$	The set of immediate predecessors of task τ_i
$EST(\tau_i, pe_j)$	The earliest execution start time of task τ_i on processor pe_j
$EFT(\tau_i, pe_j)$	The earliest execution finish time of task τ_i on processor pe_j
AFLs	Available frequency levels
DAG	Directed acyclic graph
DVFS	Dynamic Voltage Frequency Scaling
CCR	Communication to computation ratio
SLR	Scheduling Length Ratio
ECR	Energy Consumption Ratio
POF	Probability of Failure
RDLS	The Reliable Dynamic Level Scheduling algorithm
HRDS	The Hierarchical Reliability-Driven Scheduling algorithm
RHEFT	The Reliability-aware Heterogeneous Earliest Finish Time algorithm
RCPOP	The Reliability-aware Critical-Path-On-a-Processor algorithm
RMEC	The Reliability Maximization with Energy Constraint algorithm
S_n	The total number of nodes of a special DAG

heterogeneous processors. And each one is DVFS enabled. As shown in Table 2, each core can run at different AFLs. For each processor $pe_i \in PE$, the voltage-relative frequency AFLs is selected randomly among the distinct sets. The execution costs of each node on different processors are shown in Table 3, under the condition that each task runs at the maximum available frequency. According to the previous study [17], frequency switching takes a negligible amount to time, about 189–300 µs. These overheads are not taken into account in this study while applying the DVFS technique. Besides, communications among processors are also considered to perform at the same speed on all links without the limitation of bandwidth.

2.4. Fault model

While an application is executing, a fault maybe hard to avoid owing to various reasons, such as hardware failure, software bugs, devices exposed to extreme temperatures, and external interference. As a consequence, transient faults occur more frequently than permanent failures [19,28]. In this paper, we will pay more attention to transient faults in our study, and devise a feasible and efficient scheduling algorithm with the DVFS technology to maximize overall system reliability.

Extensive works have been done for fault management. Generally, the transient fault is modeled by a Poisson distribution with an average arrival rate λ [40]. Following most previous studies, we assume that transient faults happen during the execution of each task independently. Nevertheless, with the effect of dynamic voltage and frequency scaling, transient faults' average arrival rate will depend on the system processing frequency f, and v is the corresponding voltage. Hence, the fault rate can be modeled as follows:

$$\lambda(f) = \lambda_0 \cdot g(f). \qquad (4)$$

In the above equation, we have $g(f_{\max}) = 1$, where $f_{\max} = 1.0$. Traditionally, it has been recognized as an exponential relationship between the transient fault rate and the circuit's critical cost [18]. We adopt the exponential model proposed in [44] for our scheduling model and experiment analysis. It can be expressed as

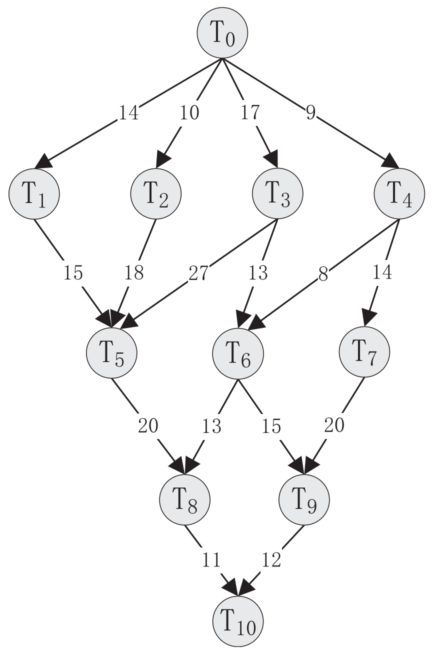

Fig. 1. A simple precedence-constrained application DAG.

Table 2
Voltage-relative frequency pairs.

Level	Pair 1		Pair 2		Pair 3	
	Voltage	Frequency	Voltage	Frequency	Voltage	Frequency
0	1.75	1.0	1.5	1.0	2.2	1.00
1	1.50	0.8	1.4	0.9	1.9	0.85
2	1.40	0.7	1.3	0.8	1.6	0.65
3	1.20	0.6	1.2	0.7	1.3	0.50
4	1.00	0.5	1.1	0.6	1.0	0.35
5	0.90	0.4	1.0	0.5		
6			0.9	0.4		

Table 3
Computation costs on different processors.

Node number	P_1	P_2	P_3
0	12	15	13
1	11	16	10
2	14	12	15
3	9	13	7
4	15	20	17
5	7	9	15
6	13	12	11
7	11	9	18
8	13	18	15
9	9	12	17
10	12	10	16

$$\lambda(f) = \lambda_0 \cdot g(f) = \lambda_0 \cdot 10^{\frac{d(1-f)}{1-f_{\min}}}, \tag{5}$$

where λ_0 stands for the average fault rate as mentioned before, d is a constant greater than zero, which represents the dependency of fault rate on frequency and voltage scaling. It can be seen easily that the fault rate will increase exponentially when the frequency decreases for energy conservation. In other words, $\lambda(f)$ is a strictly decreasing function. Hence, the maximum average fault rate is $\lambda_{\max} = \lambda_0 \cdot 10^d$, which corresponds to the minimum available frequency.

Definition 1. The reliability of a task is the probability of executing the task successfully. If the transient fault follows a Poisson distribution, the reliability of node τ_i with the corresponding computation cost c_i is [44]

$$R_i(f_i) = e^{-\lambda(f_i)\times\frac{c_i}{f_i}}, \tag{6}$$

where f_i denotes the processing frequency.

Definition 2. The system reliability R_{sys} denotes the probability of successfully executing an entire task set which consists of n tasks:

$$R_{sys} = \Pi_{i=1}^{n} R_i(f_i). \tag{7}$$

2.5. Problem description

The problem to be solved in this paper can be formally described as follows. Assume that we are given a DAG which comprises tasks with precedence constraints, and processors in a heterogeneous cluster which support different frequency levels. Then, the problem to be addressed in this paper is to assign a property execution voltage (or frequency) of an available processor for each ready task in a right order, while assuring the maximum system reliability and guaranteeing the total energy consumption not exceeding a given energy E^*. Obviously, it is a combinatorial optimization problem, whose formulation is given as follows:

$$\text{Maximize:}\ R_{sys} = \Pi_{i=1}^{n} R_i(f_i),$$

$$\text{subject to:}\ f_{\min} \leqslant f_i \leqslant f_{\max},\ (\forall\ i:\ 1 \leqslant i \leqslant n), \tag{8}$$

$$E_{total} \leqslant E^*. \tag{9}$$

3. A motivational example

As an illustration, Fig. 2 presents the schedule obtained by the RMEC algorithm for the sample DAG in Fig. 1. The schedule length is equal to 127.33, the total energy consumption is 236.70, and the probability of failure $(1 - R_{sys})$ is 2.37e−6. While the corresponding scheduling results of the RHEFT and RCPOP algorithms in terms of schedule length, total energy, and probability of failure are 138.49, 256.32, 2.57e−6, 142.49, 258.02, 2.75e−6, respectively.

4. The proposed algorithms

Reliability is critical for an application, sometimes higher system reliability is even more important than shorter schedule length. For a real application, a more useful objective is to assure that the total energy consumption does not exceed E^* and the system reliability is maximized. In a cluster, task scheduling consists of two major stages, i.e., task priority calculation

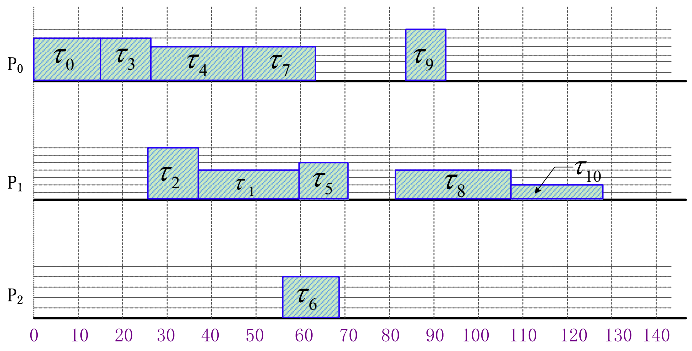

Fig. 2. Scheduling the task graph in Fig. 1 using the RMEC algorithm.

and task-processor pair selection. In the first stage, we use the efficient *URank* to calculate the task priorities, which can guarantee the topological order of a given DAG. In the second phase, there are more feasible strategies to decide the task-processor pair while fulfilling the energy constraint.

In this section, three schemes are developed to solve the problem of maximizing the reliability with energy conservation for parallel tasks in a heterogeneous cluster. The first scheme, namely the Reliability-aware Heterogeneous Earliest Finish Time (RHEFT) algorithm, is to choose the proper frequency to maximize the task execution reliability with the earliest finish time on a heterogeneous processor. The executing frequency for each task depends on the remaining energy and system reliability maximizing. The second scheme to be presented is the Reliability-aware Critical-Path-On-a-Processor (RCPOP) algorithm. In comparison with the RHEFT algorithm, the RCPOP algorithm only differs in the phase of processor selection after finding the efficient frequency for each ready task. Under such a scheme, the distinguishing feature is to find a processor with the highest reliability which can be allocated to the critical path of the application on it. In addition, a new scheme, which is referred to as the Reliability Maximization with Energy Constraint (RMEC) algorithm, picks out the best combination of task-processor according to the maximum reliability and energy conservation during the first round task scheduling. After the pre-scheduling, the available time slots are reclaimed to further reduce the energy.

Before introducing the details of the above three schemes, we firstly present the characteristics of high reliability and energy with task executing frequency.

4.1. Relationship between reliability and energy

In this section, we will present a Reliability Maximization with Energy Constraint (RMEC) algorithm. The RMEC algorithm which will be presented in the following aims at achieving high reliability with the condition of energy constraint and without increasing makespan during the scheduling procedure. Due to the frequency-independent active power, the power consumption no longer varies monotonically with the increasing of frequency. As shown in Eq. (2), it can be easily to deduce that $E_i(f_i)$ is a strictly convex function and is minimum when $f_i = f_{ee} = \sqrt[3]{P_{ind}/2C_{eff}}$ (energy efficient frequency) [41]. Therefore, lower frequencies may not always be of benefit for energy saving and there must exist an optimal voltage-frequency pair for each candidate task to achieve minimum energy consumption. On the other hand, as the processing frequency increases, the reliability of task improves monotonically. It is safely to draw such a conclusion that high reliability and low energy consumption are two contradicting elements for scheduling. Searching a good strategy for the tradeoff of them is a worthwhile work. We will attempt to schedule tasks aiming at improving system reliability and preserving energy consumption.

Definition 3. The immediate neighboring frequency f_{in} of f_{ee} on processor pe_j, is the one which consumes less energy.

According to the expression $f_i = f_{ee} = \sqrt[3]{P_{ind}/2C_{eff}}$, f_{ee} only depends on P_{ind}. The specific P_{ind} value for each task follows a uniform distribution and is determined randomly with its range between 0.2 and 2.0. So it is not difficult to find that f_{ee} varies on different processors for each task τ_i.

4.2. Critical phases

Each of the three algorithms which will be presented in the next part, is mainly composed of three phases. In the following, we introduce two important phases as outlined below.

Task priority establishment phase. To meet the requirement of task scheduling, a prior order is established in this phase. Each task is set with its *URank*, which is computed recursively according to the expression

$$URank(\tau_i) = \overline{w_i} + \max_{\tau_j \in child(\tau_i)} \left(c_{i,j} + URank(\tau_j)\right), \tag{10}$$

where $child(\tau_i)$ is the set of immediate children of task node τ_i. The rank is computed recursively by traversing from the bottom of a DAG to the top. It should be apparent to draw such a conclusion that $URank(\tau_{exit}) = \overline{w_{exit}}$. Similarly, the downward rank *DRank* is defined as

$$DRank(\tau_i) = \max_{\tau_j \in parent(\tau_i)} \left(\overline{w_j} + c_{i,j} + DRank(\tau_j)\right), \tag{11}$$

where $parent(\tau_i)$ is the set of direct parents of task node τ_i. As there is no parent for the entry node, it is easy to conclude that the *DRank* of the entry node is equal to zero. As shown in Table 3, the computation cost of each node which is performed on a special processor with the maximum available processing speed (AFL 0) is obtained. According to the computation cost given in Table 3, a priority queue for the simple DAG shown in Fig. 1 is maintained for the following three algorithms. The values of task priority using the *DRank* and *URank* method are presented in Table 4. *DRank* is also effective but less efficient than *URank* while forming a priority queue during the task priority establishment phase. Both of them can guarantee the priority constraint while scheduling. Furthermore, they are two indispensable elements to search the critical path of a DAG. In what follows, we use the *URank* to present our algorithms.

Processor frequency selection phase. As mentioned above, the "best" frequency-voltage pair to achieve lower energy and higher reliability appears as nearest neighbors of f_{ee} on a processor for each task node. A binary search tree is used to find these two immediate neighbors, which has time complexity of $O(\log|T|)$, where $|T|$ is the number of task nodes. By calculating the active energy using Eq. (2), the neighbor frequency which consumes less energy will be selected to perform a data ready task node.

4.3. The proposed algorithms

4.3.1. The Reliability-aware Heterogeneous Earliest Finish Time (RHEFT) Algorithm

For the purpose of performance comparison, two other revised algorithms which are based on two well known algorithms will be presented in order to find the optimal system reliability. The first revised algorithm is the Reliability-aware Heterogeneous Earliest Finish Time (RHEFT) algorithm.

Algorithm 1. RHEFT

```
Require: A DAG $G = \langle T, E\rangle$ and a set $PE$ of DVS available processors.
Ensure: A schedule $S$ of $G$ onto $PE$.
 1: compute $URank$ of $\tau_i \in T$ by traversing the graph from the exit node
 2: compute energy-efficient frequency for each node in set $T$
 3: sort the tasks in a non-increasing order by $URank(\tau_i)$ value and establish a priority queue $Queue_{URank}$ for the sorted
    tasks
 4: while the priority queue $Queue_{URank}$ is not empty do
 5:    $\tau_i \leftarrow$ the head node in $Queue_{URank}$
 6:    for each processor $pe_j \in PE$ do
 7:       for $\forall f_{j,k} \in F$ do
 8:          find the immediate neighboring frequency $f_{in}$ of $f_{ee}$ on processor $pe_j$ for task $\tau_i$, mark the one which consumes
             less energy
 9:          if the summation of energy consumption satisfies $E_{total} \leqslant E^*$ then
10:             $f_{j,k} \leftarrow f_{in}$
11:          else
12:             $f_{j,k} \leftarrow f_{max}$
13:          end if
14:       end for
15:    end for
16:    assign the marked frequency $f_{in}$ to task $\tau_i$ on the marked processor
17:    compute the reliability of task $\tau_i$ using Eq. (6)
18:    compute the energy consumption for task $\tau_i$ using Eq. (2)
19:    delete the head node $\tau_i$ in $Queue_{URank}$
20: end while
```

The RHEFT algorithm (Algorithm 1) is an application scheduling algorithm with bounded number of heterogeneous processors. It consists of three major phases, i.e., the priority establishment phase to provide a valid topological order of application tasks, the frequency selection phase to choose a feasible and efficient frequency to perform the data ready task, and the processor assignment phase to allocate the candidate task to the "best" processor in the order of priorities and low energy consumption.

As shown in Algorithm 1, Step 1 and Step 2 primarily calculate the *URank* and energy-efficient frequency for each task, respectively. The priority queue is constructed in Step 3. Steps 4–20 are outside loop of the RHEFT to guarantee each task node is scheduled in a right order. In Step 8, a suitable immediate neighboring frequency is selected, and temporarily marked in the memory. Steps 9–13 guarantee that the total energy consumption does not exceed E^*. After traversing all the processors in Steps 6–15, the most suitable frequency and the "best" processor are produced. Then the algorithm allocates a task node to the best processor and assigns it with suitable execution frequency f_{in} in Step 16. The reliability and energy consumption of a task are calculated in Steps 17 and 18, respectively. The RHEFT algorithm takes $O(\log f)$ time for the frequency selection phase in Step 8. The RHEFT algorithm has a $O(|T| \times p \times \log f)$ time complexity for $|T|$ task nodes and p processors, where each processor supports f levels of DVS enabled frequencies.

4.3.2. The Reliability-aware Critical-Path-On-a-Processor (RCPOP) Algorithm

Another revised algorithm is the Reliability-aware Critical-Path-On-a-Processor (RCPOP) algorithm. In this algorithm, the *DRank* and *URank* of each node is computed firstly. A priority queue of the application graph is formed according to the decreasing order of nodes' *URank* values. The task nodes on the critical path, which has the maximum summation of *DRank* and *URank*, should be found and stored to a sorted list. Then in Step 9, energy-efficient frequency f_{ee} is derived. The critical processor which takes the least computational cost for all task nodes on the critical path is selected in Step 10. Steps 12–28 are the main part of a loop. While scheduling a task from the priority queue, a node on the critical path will only be assigned to the critical processor. Then the algorithm picks out a "best" immediate neighboring frequency f_{in} for it under the constraint of total computation consumption. The nodes in other paths can also be assigned to the critical processor provided that there is a free and big enough slack time slot to accommodate it. Otherwise, it will be assigned to a non-critical processor. The latter procedure is the same as the CP nodes. For all ready tasks, the feasible frequency searching takes time $O(\log f)$. So the complexity of the RCPOP algorithm is $O(|T| \times p \times \lg f)$.

Algorithm 2. RCPOP

```
Require: A DAG $G = \langle T, E \rangle$ and a set $PE$ of DVS available processors.
Ensure: A schedule $S$ of $G$ onto $PE$.
 1: compute $URank(\tau_i)$ of each node $\tau_i \in T$ by traversing the graph upward from the exit node
 2: compute $DRank(\tau_i)$ of each node $\tau_i \in T$ by traversing the graph downward from the entry node
 3: $|CP| = Max\{\forall \tau_i \in T | URank(\tau_i) + DRank(\tau_i)\}$
 4: for each node $\tau_i \in T$ do
 5:   if the summation of $URank(\tau_i)$ and $DRank(\tau_i)$ equals to $|CP|$ then
 6:     add node $\tau_i$ to $CP$ set $list_{CP}$
 7:   end if
 8: end for
 9: compute energy-efficient frequency for each node in set $T$
10: select the CP processor $P_{CP}$ which have the minimal summation computation cost of nodes in $list_{CP}$
11: sort the tasks in a non-increasing order by $URank(\tau_i)$ value and establish a priority queue $Queue_{URank}$ for tasks in
    order
12: while not all nodes in $Queue_{URank}$ have been scheduled do
13:   $\tau_i \leftarrow$ the head node in $Queue_{URank}$
14:   if $\tau_i \in list_{CP}$ then
15:     assign processor $P_{CP}$ to $\tau_i$, and compute the immediate neighboring frequency $f_{in}$ of $f_{ee}$ on processor
16:   else
17:     assign the processor which consumes less energy with immediate neighboring frequency of $f_{in}$ its $f_{ee}$ for $\tau_i$
18:   end if
19:   if the summation of energy consumption satisfies $E_{total} \leqslant E^*$ then
20:     $f_{j,k} \leftarrow f_{in}$
21:   else
22:     $f_{j,k} \leftarrow f_{max}$
23:   end if
24:   assign task $\tau_i$ to the marked processor and specify its executing frequency $f_{in}$
25:   compute reliability of the task node $\tau_i$ using Eq. (6)
26:   compute energy consumption of the task node $\tau_i$ using Eq. (2)
27:   delete the head node $\tau_i$ in $Queue_{URank}$
28: end while
```

4.3.3. The Reliability Maximization with Energy Constraint (RMEC) Algorithm

Likewise, three phases are incorporated into the RMEC algorithm. The task priority queue consists of Steps 1 and 2. The topological order of tasks is established in this phase. The optimal frequency selection phase comprises Steps 5–13. It is notable that Steps 7–11 assure the RMEC algorithm within the energy constraint, as well as improve the reliability of each candidate task during execution. It is in the inner loop body of the algorithm. Inspired by the energy-conscious scheduling method proposed in [23], the reliability maximum energy conservative (RME) strategy is to choose a processor and frequency combination that has the maximum value, as shown in Eq. (12),

$$\mathrm{RME}(\tau_i, pe_j, f_{j,k}, pe_l, f_{l,m}) = -\left(\frac{R(\tau_i, pe_j, f_{j,k}) - R(\tau_i, pe_l, f_{l,m})}{R(\tau_i, pe_j, f_{j,k})}\right) + \left(\frac{E(\tau_i, pe_j, f_{j,k}) - E(\tau_i, pe_l, f_{l,m})}{E(\tau_i, pe_j, f_{j,k}) - \min\{E(\tau_i, pe_j, f_{j,k}) - E(\tau_i, pe_l, f_{l,m})\}}\right), \tag{12}$$

where τ_i is the data ready task which is in the head of the priority queue, $f_{j,k}$ and $f_{l,m}$ are the available discrete frequency on the candidate processors pe_j and processor pe_l, respectively. $R(\tau_i, pe_j, f_{j,k})$ and $R(\tau_i, pe_l, f_{l,m})$ denote the reliability of task τ_i on processor pe_j at frequency $f_{j,k}$ and that of task τ_i on processor pe_l at frequency of $f_{l,m}$, respectively. Similarly, $E(\tau_i, pe_j, f_{j,k})$ and $E(\tau_i, pe_l, f_{l,m})$ represent the energy consumption of task τ_i on processor pe_j at frequency $f_{j,k}$ and that of task τ_i on processor pe_l at frequency of $f_{l,m}$, respectively.

Algorithm 3. RMEC

```
Require: A DAG $G = \langle T, E \rangle$ and a set $PE$ of DVS available processors.
Ensure: A schedule $S$ of $G$ onto $PE$.
1: compute $URank(\tau_i)$ of each node in DAG $G$
2: sort task $T$ in a non-increasing order by $URank(\tau_i)$ to establish priority queue $Queue_{URank}$
3: while the priority queue $Queue_{URank}$ is not empty do
4:    $\tau_i \leftarrow$ the head node in $Queue_{URank}$
5:    for each processor $pe_j$ in set $P$ do
6:       compute the nearest frequency level $f_{in}$ of processor $pe_j$ which has the minimal energy consumption
7:       if the summation of energy consumption satisfies $E_{total} \leqslant E^*$ then
8:          $f_j \leftarrow f_{in}$
9:       else
10:         $f_j \leftarrow f_{max}$
11:      end if
12:      search the best combination of processor $pe_j$ and frequency level $f_k$ of which keeps the max value
         $REM(\tau_i, pe_j, f_k, pe_t, f_t)$   ▷ $pe_t$ and $f_t$ are used to temporarily store the best combination of processor and
         processing frequency respectively
13:   end for
14:   assign the recorded best combination of processor $pe_t$ and processing frequency $f_t$ to node $\tau_i$
15:   compute the node reliability of task $\tau_i$
16: end while
17: let $S'$ denotes the scheduling derived from the above procedure
18: while the scheduling list $S'$ is not empty do
19:   $\tau_i' \leftarrow$ the head node in scheduling list $S'$
20:   for each processor $pe_j'$ in set $PE$ do
21:      while there is an available slack slot in the processor which satisfies the precedence priority of tasks do
22:         compute makespan, node reliability
23:         if makespan does not increase, node reliability promotes and there is an available time slot to accommodate
            $\tau_i'$ then
24:            replace $\tau_i'$ with the better combination processor $pe_j'$ and frequency $f_k'$
25:         end if
26:         update node reliability and compute the node energy consumption
27:      end while
28:   end for
29: end while
```

Steps 14–15 are included to realize the third phase, i.e., assigning a node in the marked processor with the optimum frequency. As slack occurs inevitably due to the inter-processor communication for a DAG, Steps 17–29 are used to utilize the slot times of processors without increasing the schedule length and degrading reliability. This step can be taken as local optimum without sacrificing time complexity consumption. So the complexity of the RMEC algorithm is $O(2 \times |T| \times p \times \log f)$.

5. Experiments and results

RHEFT and RCPOP are two algorithms with straightforward minds for reliability maximization. In the comparison with the RMEC algorithm, they are inferior in performance. Hence, in this section, we not only compare the three proposed algorithms, but also compare the performance of the RMEC algorithm with two other excellent reliability-aware algorithms, i.e., HRDS [30] and RDLS [10]. These two algorithms are proven to perform well for parallel task scheduling in a heterogeneous computing system. The following comparison is intended not only to present quantitative results, but also to qualitatively analyze the results. To measure the performance of these scheduling algorithms, two extensive sets of task graphs, i.e., randomly generated and real-world applications, are used in our study. The three real parallel applications are the fast Fourier transformation (FFT), the LU-decomposition, and Gaussian-elimination. In addition, plenty of variables are made for these task sets in some more comprehensive cases.

Before making a comparison with algorithms RDLS and HRDS, we firstly give a brief introduction of them. HRDS [30] refers to the Hierarchical Reliability-Driven Scheduling algorithm in a hierarchical grid computing system. Its main objective is to achieve high reliability and shorter schedule length by considering the task reliability overhead while mapping a task to the appropriate processor. The algorithm proceeds in two phases. During the first phase of establishing task priority, the system task overhead $LC(\tau_j, pe_j)$ is taken into account, where $LC(\tau_j, pe_j)$ is the sum of the task reliability cost and the corresponding earliest finish time of task τ_j on processor pe_j. During the second phase, the algorithm selects the processor pe_m which costs the minimal overhead $LC(\tau_j, pe_j)$ to perform task τ_j. That is to say, for a given task τ_i in the task set T, HRDS allocates the processor pe_m with minimal $LC(\tau_j, pe_m)$ in the mapping phase of list scheduling.

While the Reliable Dynamic Level Scheduling (RDLS) algorithm [10] is based on the DLS [9], it promotes the best suited resources to the application to reduce the execution time of that application. During the task priority calculation phase, RDLS adds the last item $C(\tau_i, pe_j)$ to capture the reliability, which could impact the allocation decision and improve the reliability of application. It is noteworthy that the $C(\tau_i, pe_j)$ is a cost function, which could promote resources reliability at a high level to maximize the reliability of an application. Akin to the DLS algorithm, the RDLS algorithm finds a suitable task-processor pair with the highest dynamic level. The RDLS algorithm minimizes the makespan and enhances the reliability of the application to some extent.

Specially, the random task graph set consists of 100 base task graphs generated with 80 different sizes, eleven CCRs, and five processor heterogeneity settings. For each configuration, the specific P_{ind} value is determined randomly according to a uniform distribution in the range of [0.2, 2.0]. For simplicity, the value of C_{eff} is set to one throughout the whole experiments. Each of the three real-world applications is conducted in the benchmark that involves 7250 task graphs. Hence the total amount is 21,750.

Our experiments are carried out using a workstation at National Supercomputing Center in Changsha. It is equipped with an Intel Core i3-540 dual-core CPU, 8 GB DRAM, and 500 GB hard disk, respectively. The machine runs with Windows 7 (64 bit OS) SP1. We designed a simulator which has implemented well known algorithms, including HCPFD [14], HLD [3], RDLS [10], and so on. Based on this framework, we utilize the well-known benchmark involving the large number of FFT, GE, Laplace (introduced below) graphs to implement our proposed algorithms to verify the performance.

In our study, each processor is assumed to execute a task from the prior queue without preemption. The computation and communication costs are generated and follow a uniform distribution for each task node, with the mean value to the specific average cost of computation and communication, respectively. The occurrence of transient fault follows a Poisson

Table 4
Task priorities.

Node number	*URank*	*DRank*
0	136.33	0.00
1	96.67	27.33
2	101.00	22.33
3	106.00	30.33
4	101.33	22.33
5	69.33	67.00
6	64.33	53.00
7	70.00	53.67
8	39.00	97.33
9	37.33	86.33
10	12.67	123.67

distribution, and the equation is given by Eq. (5), where $d = 3$ and $\lambda_0 = 10^{-9}$. For the three real applications, the number of tasks can range from about 14 to 1024 as the input number of points and the matrix sizes are varied, respectively.

5.1. Performance metrics

5.1.1. Scheduling Length Ratio (SLR)

Makespan, or scheduling length, is defined as

$$\text{makespan} = FT(\tau_{exit}), \tag{13}$$

where $FT(\tau_{exit})$ is the earliest finish time of the exit task in the scheduling queue.

For most scheduling algorithms, makespan is one of the main metrics of performance evaluation. Since a large set of application graphs with different properties are used, it is necessary to normalize the makespan to the lower bound, which is named as Scheduling Length Ratio (SLR). Without considering the communication cost, the calculation expression of SLR can be expressed as:

$$\text{SLR} = \frac{\text{makespan}}{\sum_{\tau_i \in CP} \min_{pe_j \in PE}\{w_{i,j}\}}. \tag{14}$$

5.1.2. Energy Consumption Ratio (ECR)

In our study, we consider energy consumption and the probability of failure for a scheduled task set as another two more important performance metrics. For a given task set, the ECR is defined as:

$$\text{ECR} = \frac{E_{total}}{\sum_{\tau_i \in CP} \min_{pe_j \in PE}\left\{P_{ind_i} \cdot \frac{c_i}{f_i} + C_{ef} \cdot c_i \cdot f_i^2\right\}}, \tag{15}$$

where E_{total} is the total energy consumption of the scheduled tasks.

5.1.3. Probability of Failure (POF)

The probability of failure is one of the two primary performance metrics for our comparison. Formally, the POF value of a task set with the allocated frequency to each specific task by a scheduling algorithm is defined as:

$$\text{POF} = 1 - R = 1 - \Pi_{i=1}^{n} R_i(f_i). \tag{16}$$

5.2. Randomly generated DAG

In our study, we first considered the random DAG graphs, which are generated with certain probability for an edge between any two nodes of the graph. Such kind of weighted application DAGs with various characteristics that have close relationship with several input parameters can be presented briefly as follows.

- DAG size (τ). This is the number of task nodes in a graph.
- Communication to computation ratio (CCR). In some experiments, CCR is used to characterize the workload of the task graph. It is the ratio which equals to the average of communication cost over the average of computation cost. With the help of CCR, one can judge the importance of communication or computation in the task graph. A DAG with very low value can be considered as a computation intensive application. On the contrary, a DAG with a high CCR value is a communication intensive application.
- Average out degree (δ). This is the average value of the out degree of a DAG graph.
- Computation capacity heterogeneity factor (h) [8]. Heterogeneity basically reflects the variance of processing speed. A high h indicates wider margin in the computation costs for a task, and vice versa. And each task has the same computation cost if this factor is set to 0. The average computation cost of each task $\overline{w(\tau_i)}$ is selected randomly from a uniform distribution generator with a mean value W which can be specified by a user. And its range is $\left[\overline{w(\tau_i)} \times (1 - \frac{h}{2}), \overline{w(\tau_i)} \times (1 + \frac{h}{2})\right]$. The value of W has no influence on the performance of scheduling.

In each of our experiments, graphs are created with the combination of the above characteristics. The graph size ranges from 50 to 500 nodes, with step by 50. The communication edge between two nodes is generated with identical probability and computed based on the average number of edges for each node.

5.3. Random application performance results

The goal of these experiments is to assess the performance of the proposed algorithms with two other excellent algorithms, i.e., HRDS and RDLS.

The experimental results of the first set of simulation studies are shown in Figs. 3–5, where each data point comes from the average value of experiment data based on the algorithms running for 500 times. As can be seen from these three figures, RMEC outperforms the other four algorithms significantly for applications with different sizes and diverse CCR values. For Fig. 3, where the application is computation intensive with CCR = 0.5, the average SLR for both RMEC and HRDS are pretty close. This is due to the fact that SLR is not the most important metric in this study. To achieve high system reliability even at the expense of modest performance loss in some cases, RMEC tries its best to improve the reliability of a task with the minimum energy consumption to the greatest extent. It can also be observed from Fig. 3 that RMEC outperforms HRDS and RDLS in terms of average ECR and average POF. We attribute the marginally better performance of RMEC over other four algorithms to the fact that RMEC is a reliability adaptive strategy and assigns each task intelligently to a processor with an appropriate execution frequency according to its computational time and execution reliability. In the processor allocation

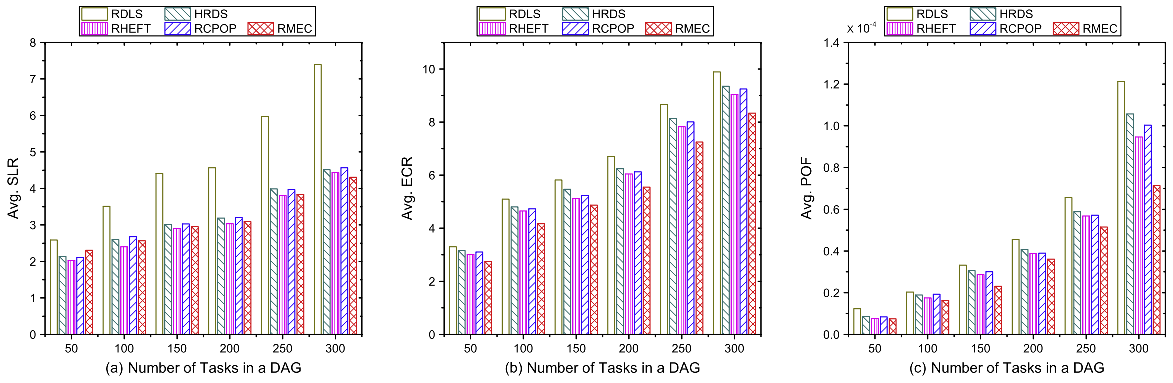

Fig. 3. Average SLR, ECR, and POF of the five algorithms for CCR = 0.5.

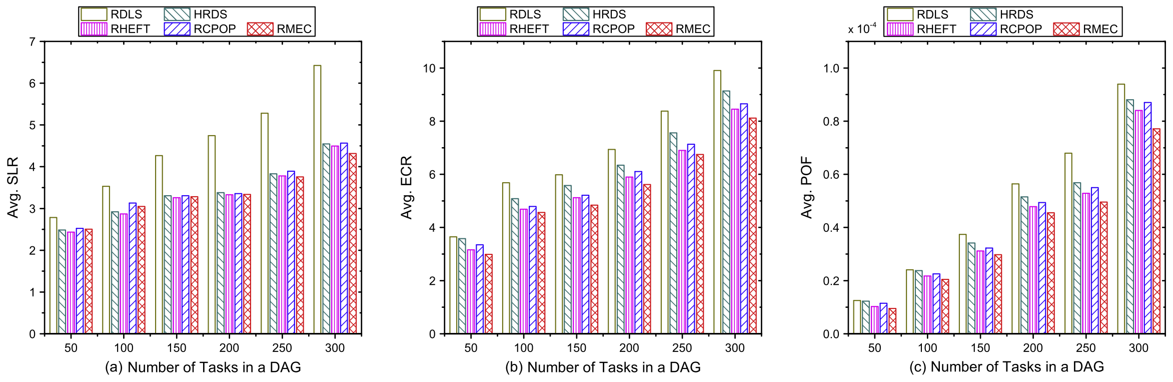

Fig. 4. Average SLR, ECR, and POF of the five algorithms for CCR = 1.0.

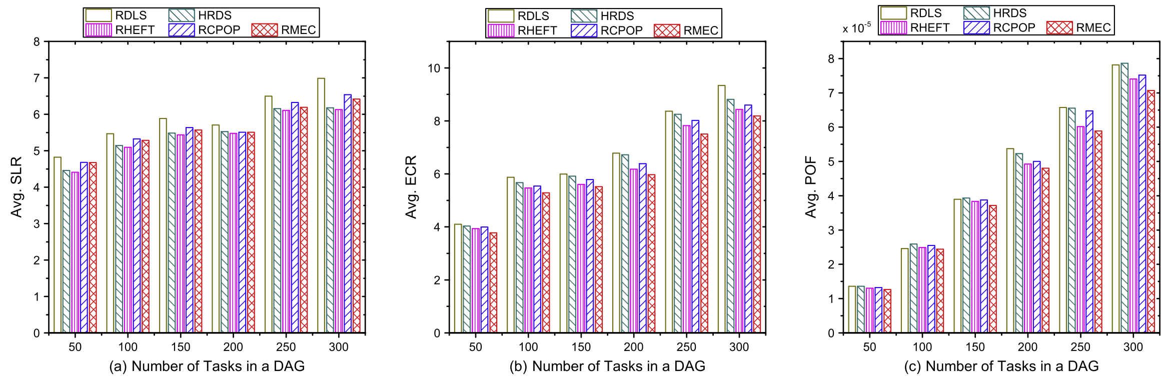

Fig. 5. Average SLR, ECR, and POF of the five algorithms for CCR = 5.0.

phase, RHEFT chooses the processor with the earliest finish time, and RCPOP identifies the critical processor for critical tasks. Both of them do not take enough account for reliability. Meanwhile, HRDS pays excessive attention to system task overhead $LC(\tau_j, pe_j)$ in the phase of processor selecting. And the cost function of RDLS cannot accurately reflect the importance of the efficient frequency selection for an application's reliability. Thus, their ECR and POF are all worse than RMEC.

The comparisons for random graphs with larger CCR values are shown in Figs. 4 and 5, respectively. The gaps among the five algorithms shrink gradually with the CCR value increasing. For both CCR = 1.0 and CCR = 5.0, the improvement of RMEC evinces significantly, with respect to the quality of average ECR and average POF. Specially, as CCR equals to five, it implies that the cost of communication part dominates the whole application. RMEC clearly exceeds RHEFT, RCPOP, HRDS, and RDLS by (3.31%, 5.74%, 7.62%, 14.05%) in terms of the average ECR, and (3.14%, 6.22%, 11.23%, 14.05%) in terms of the average POF, respectively. With respect to average SLR, RMEC has slightly lower performance (−2.95%, 1.10%, −3.68%, 3.89%) over RHEFT, RCPOP, HRDS, and RDLS. RMEC is a reliability-aware and energy conservative algorithm. It obtains nice ECR and POF at the expense of slight longer makespan. The results distinctly demonstrate that RMEC surpasses HRDS and RDLS. For the above

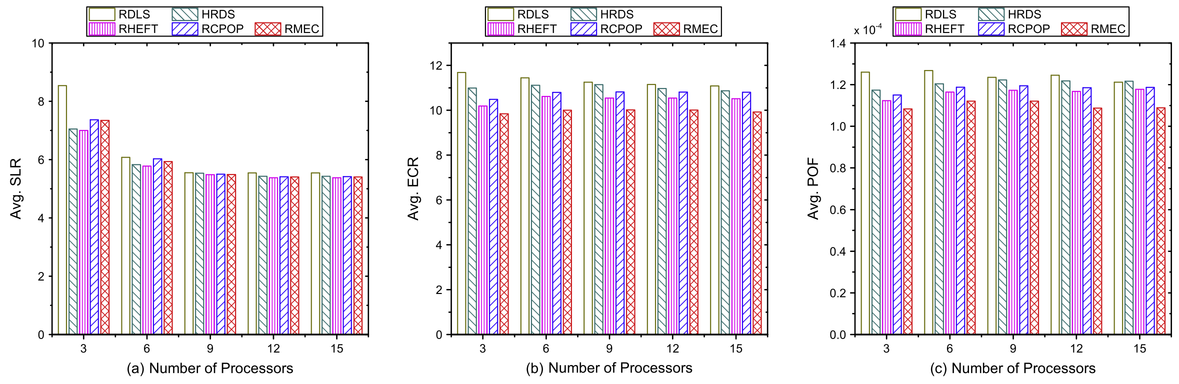

Fig. 6. Average SLR, ECR, and POF of the five algorithms for CCR = 5.0 and DAG size = 500.

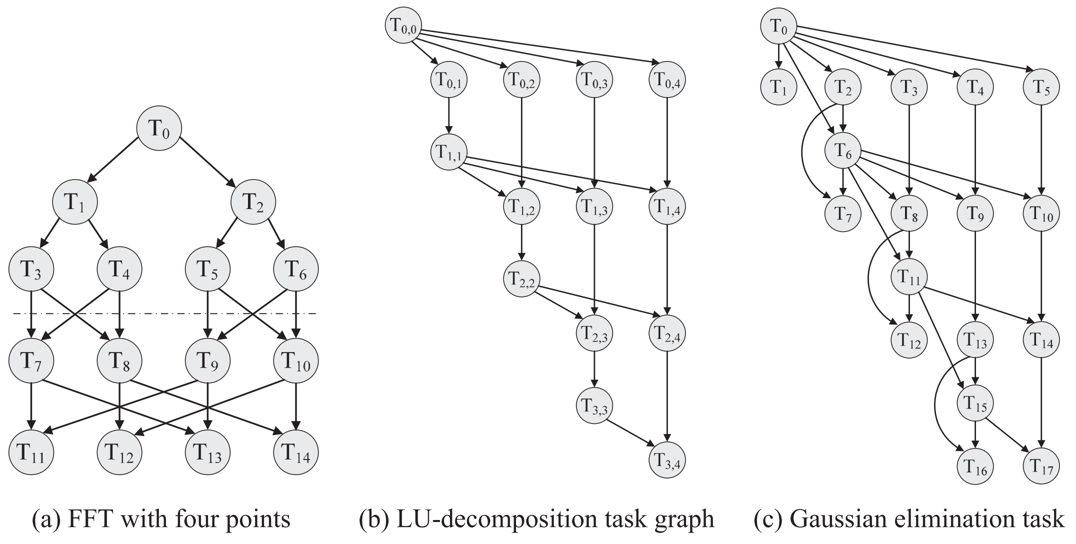
(a) FFT with four points
(b) LU-decomposition task graph
(c) Gaussian elimination task

Fig. 7. Three kinds of real-world DAGs.

Table 5
Configuration parameters for the FFT task graphs.

Parameter	Possible values
CCR	0.5, 1, 2, 3, . . . , 10
Number of processors	3, 6, 9, 12, 15
Number of points	4, 8, 12, 16, 20, 24

analysis, we can get a conclusion that our algorithms perform better as the CCR increases, especially the RMEC algorithm. In other words, it is more suitable for data intensive applications.

Fig. 6 reveals the performance of the five algorithms for various numbers of processors, where the random graph has 500 task nodes and CCR equals to five. The RMEC strategy performs consistently better than other four algorithms with the number of processors increasing. The margins of the average SLR among the five algorithms decrease gradually. As the number of processors increases, the average of ECR and the average of POF show a slight downward trend. At some point, especially for 12 processors in Fig. 6, the increase in the number of processors is of no use to improve the performance.

5.4. Performance analysis on graphs of real-world applications

Without loss of generality, it is necessary to use real-world applications to evaluate the performance of our algorithms. To this end, three representative real-world applications will be used to test the performance of our algorithms. These three applications are real-world problems, i.e., fast Fourier transformation (FFT), LU decomposition, and Gaussian elimination.

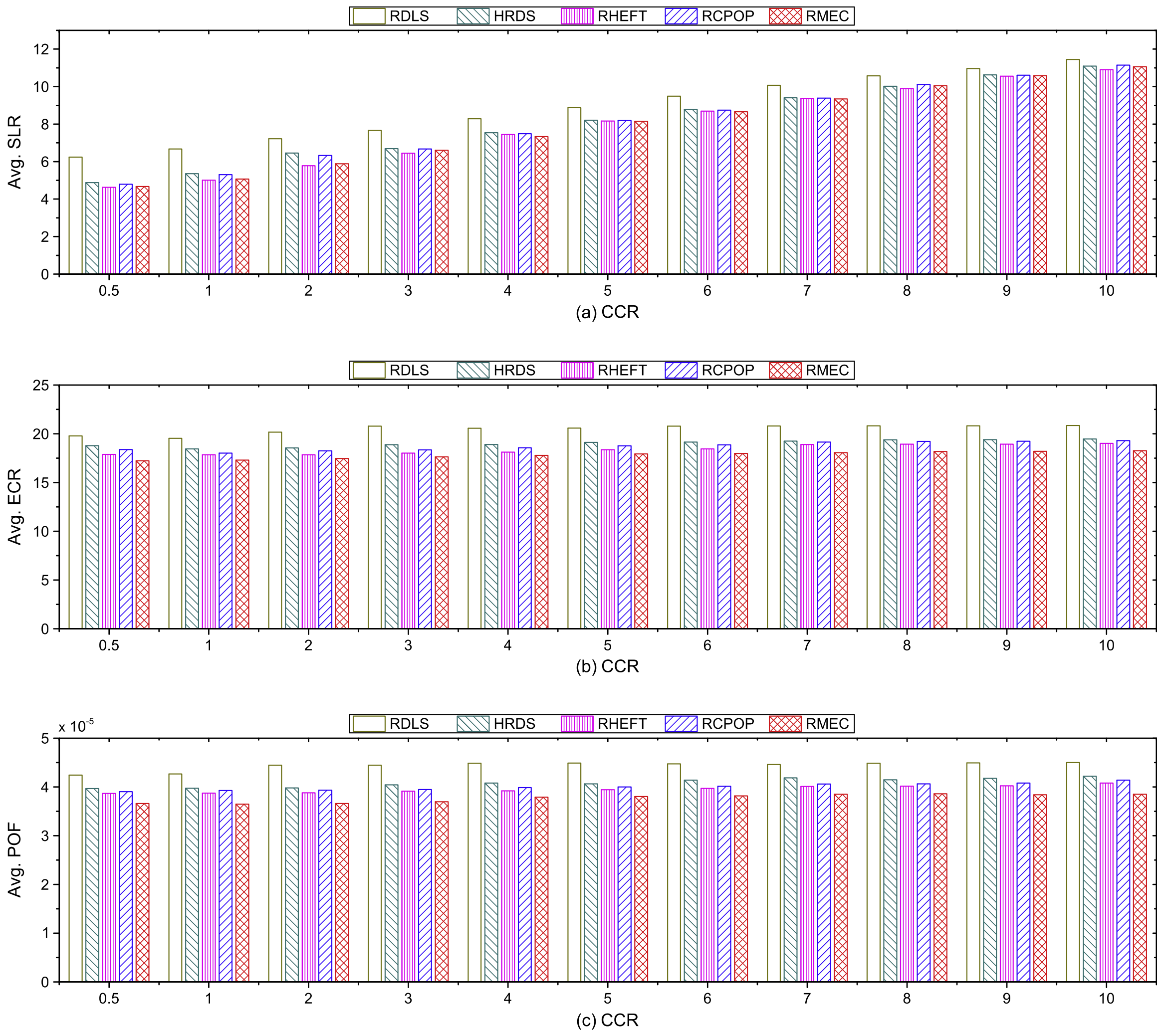

Fig. 8. Average SLR, ECR, and POF for the FFT task graph with different CCR.

Table 6
Configuration parameters for the LU decomposition task graphs.

Parameter	Possible values
CCR	0.5, 1, 2, 3, ..., 10
Number of processors	3, 6, 9, 12, 15
Size	5, 6, 7, ..., 29, 30

5.4.1. Fast Fourier transformation (FFT)

A fast Fourier transform is an algorithm to compute the discrete Fourier transform and its inverse transform. FFT computations are very fast and widely used in many applications in science, engineering, and mathematics. The computation of FFT is comprised of two operations, i.e., the input vector and the butterfly operation. Fig. 7(a) shows a FFT task graph with four points. It consists of two parts. The tasks above the dashdotted line are the recursive call tasks and the ones below are the butterfly operation tasks [8]. The parameters used for experiment are shown in Table 5. The total number of tasks used in the evaluation follows the expression, $S_{2n} = 2^{2n}, \ (n \geqslant 2)$, where n is the number of levels in a FFT graph. As the size of n ranges from 4 to 10 with the step by 1, the input FFT points change from 4 to 24 with the step by 4, and the corresponding number of task nodes in a FFT graph varies as 16, 32, 64, 128, 256, and 512. For each kind of configuration, which combines the three parameters in Table 5 together, we test the five Algorithms 500 times in such a combination. The ultimate results are evaluated with the selected average value from the output, involving the average value of SLR, ECR, and POF with the increment of CCR.

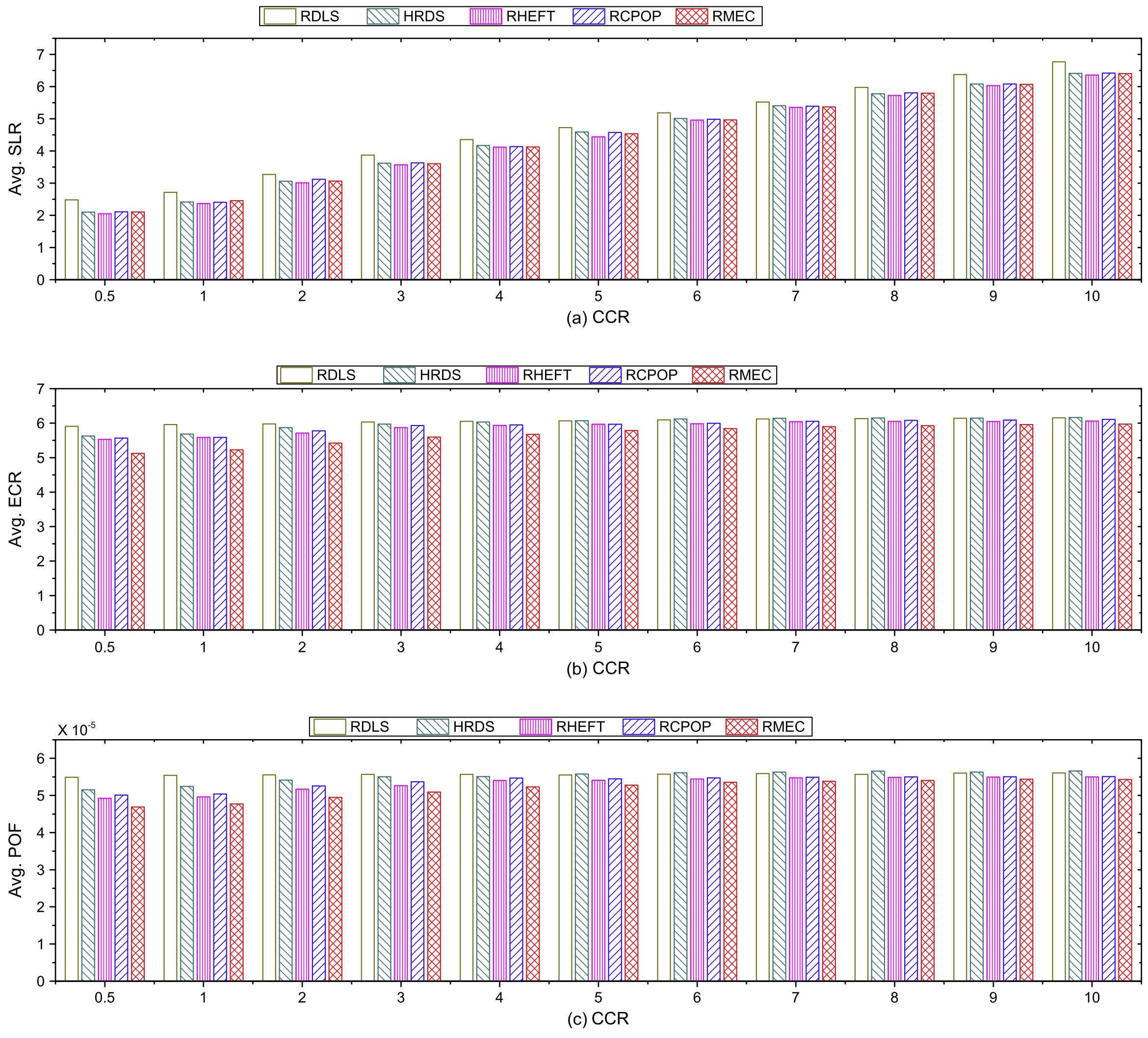

Fig. 9. Average SLR, ECR, and POF for the LU task graph with different CCR.

Table 7
Configuration parameters for the Gaussian elimination task graphs.

Parameter	Possible values
CCR	0.5, 1, 2, 3, ..., 10
Number of processors	3, 6, 9, 12, 15
Size	5, 6, 7, ..., 29, 30, 31

The experiment result are presented in Fig. 8(b) and (c). As can be observed from these two figures, algorithms RHEFT, RCPOP, and RMEC are able to produce competitive ECR and POF over HRDS and RDLS. Due to the comprehensive precedence constraints of most task nodes as the size of graph grows, the variety of both ECR and POF for the three algorithms are not large. In such a case, an increment of number for the processor will not bring any benefit for improving performance.

The overall performance improvement of the RMEC strategy for FFT graphs is 4.87%, 6.19%, 8.43%, 17.64% better than RHEFT, RCPOP, HRDS, and RDLS respectively, in terms of the average POF. For the average ECR, RMEC performs 3.21%, 5.12%, 6.79%, 15.02% better than RHEFT, RCPOP, HRDS, and RDLS. For the average SLR, RMEC performs −0.60%, 1.55%, 1.87%, 11.54% better than RHEFT, RCPOP, HRDS, and RDLS.

5.4.2. LU decomposition

LU decomposition, as shown in Fig. 7(b), is used wisely in solving mathematical equations and works by transforming a matrix into a production of lower and upper triangular matrices. Table 6 shows the used parameters for experiment of LU decomposition task graphs. The total number of task nodes S_n complies with the expression, $S_n = \frac{n^2+3n}{2}, (n \geqslant 3)$, where n is the size. With the size of input matrix varies from 5 to 30, the task node number changes from 20 to 495. With regard to the number of processors and the value of CCR, they vary from 3 to 15 with step by 3, and 0.5, 1, 2, 3, 4, ..., 9, 10, respectively. During the experiments, we choose the combination of the three parameters in Table 6, and make the algorithms execute 500 times under the condition of every combination. The following result is collected from the average value of the output.

As shown in Fig. 9(b) and (c), the RMEC strategy outperforms HRDS and RDLS strategies for all values of the CCR parameter. For both small and large CCR values, the RMEC algorithm consistently produces very low ECR values, especially for small CCR values. The overall performance improvement of the RMEC strategy for LU decomposition graphs is 2.67%,

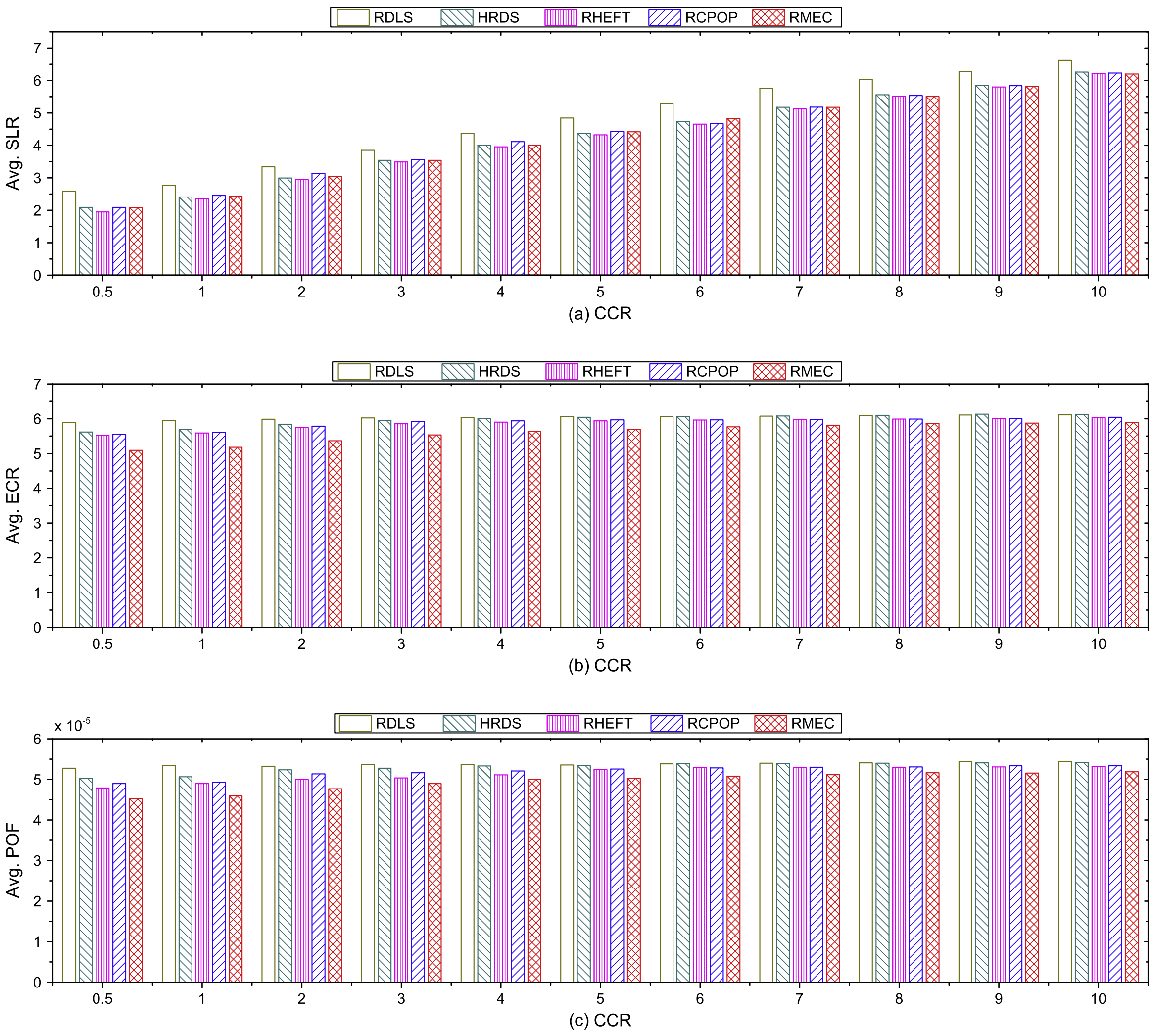

Fig. 10. Average SLR, ECR, and POF for the GE task graph with different CCR.

3.59%, 7.37%, 6.25% better than RHEFT, RCPOP, HRDS, and RDLS respectively, in terms of the average POF. For the average ECR, RMEC performs 3.77%, 4.29%, 5.69%, 6.74% better than RHEFT, RCPOP, HRDS, and RDLS. For the average SLR, RMEC performs −1.04%, 3.34%, 0.23%, 5.68% better than RHEFT, RCPOP, HRDS, and RDLS.

5.4.3. Gaussian-elimination

The Gaussian-elimination, a well known method, is widely used in mathematics for solving system of linear equations. Let n be the size of a matrix which depicts the Gaussian-elimination task graph. The total number of tasks S_n in this DAG is equal to $\frac{n^2+n-2}{2}$, $(n \geqslant 2)$, where n is the size of a matrix in a GE graph. Fig. 7(c) shows a Gaussian elimination graph with matrix size 5. Table 7 shows the corresponding parameters used in the experiment of Gaussian elimination graphs. As the size of a matrix varies from 5 to 31 with an increment step by 1, the total number of task nodes ranges from 14 to 495. Similarly as above, after 500 times execution under the combination of the there parameters in Table 7, the final comparison for the five algorithms with the average SLR, ECR, POF for different CCR is given below.

It can be seen from Fig. 10(b) and (c) that the RHEFT, RCPOP, and RMEC strategies surpass the HRDS and RDLS scheduling strategies for all the CCR values, in terms of the average ECR and the average POF. As CCR increases, the average ECR for each of the five algorithms grows slightly. The overall performance of the RMEC algorithm for the Gaussian elimination graph is 3.81%, 4.84%, 6.94%, 8.40% better than RHEFT, RCPOP, HRDS, and RDLS respectively, in terms of the average POF. For the average ECR, the RMEC performs 4.53%, 4.96%, 6.38%, 7.63% better than RHEFT, RCPOP, HRDS, and RDLS, respectively. For the average SLR, RMEC performs −1.53%, 3.77%, −1.31%, 9.96% than RHEFT, RCPOP, HRDS, and RDLS, respectively.

6. Conclusion

This paper aims at incorporating task reliability while applying the DVS technique to achieve low power consumption in a heterogeneous computing system of a cluster. Since most traditional researches lack for the consideration of transient failure while exploiting the DVS technique, we believe that it is mandatory to devise and implement highly reliable scheduling heuristics to satisfy the requirement of precedence constrained tasks, while pursuing high performance as well as achieving energy efficiency. Thus, we design scheduling algorithms that can incorporate task reliability enhancement and energy saving for directed acyclic graphs. They can achieve high system reliability.

The performance of the RMEC algorithm is compared with four other reliability-aware algorithms. All of them take proper consideration of improvement for system reliability in a multiprocessor computing system. The comparison is based on a large number of randomly generated DAGs and three real-world applications, which include fast Fourier transformation (FFT), Gaussian elimination, and LU-decomposition. The simulation results show that the proposed RMEC algorithm significantly surpasses other algorithms in terms of system reliability and energy consumption.

Robustness, high system reliability, and low energy consumption are crucial to a heterogeneous computing system in a cluster. One planned future research is to detect and recover a failed task while scaling frequency to execute tasks. This extension may come up with one or more shared recovery blocks in each processor.

Acknowledgments

The authors would like to express their gratitude to the anonymous reviewers for their insightful comments. The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), the National Natural Science Foundation of China (Grant Nos. 61370095, 61472124), the National Science Foundation for Distinguished Young Scholars of Hunan (Grant No. 12JJ1011), and the Innovation Fund Designated for Graduate Students of Hunan Province (No. CX2013B142).

References

[1] M. Abd-El-Barr, F. Gebali, Reliability analysis and fault tolerance for hypercube multi-computer networks, Inf. Sci. 276 (2014) 295–318.
[2] S. Albers, Energy-efficient algorithms, Commun. ACM 53 (5) (2010) 86–96.
[3] S. Bansal, P. Kumar, K. Singh, Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs, J. Parallel Distrib. Comput. 65 (4) (2005) 479–491.
[4] L. Benini, A. Bogliolo, G. De Micheli, A survey of design techniques for system-level dynamic power management, Trans. VLSI Syst. 8 (3) (2000) 299–316.
[5] M.A. Beyer, D. Laney, The Importance of ‘big data’: A Definition, Gartner, Stamford, CT, 2012.
[6] T.D. Burd, R.W. Brodersen, Energy efficient CMOS microprocessor design, in: Proc. of the Twenty-Eighth Hawaii International Conference on System Sciences, 1995, pp. 288–297.
[7] R.S. Chang, C.Y. Lin, C.F. Lin, An adaptive scoring job scheduling algorithm for grid computing, Inf. Sci. 207 (2012) 79–89.
[8] M.I. Daoud, N. Kharma, A high performance algorithm for static task scheduling in heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 68 (4) (2008) 399–409.
[9] A. Dogan, F. Özgüner, Optimal and suboptimal reliable scheduling of precedence-constrained tasks in heterogeneous distributed computing, in: Proc. of International Workshops on Parallel Processing, 2000, pp. 429–436.
[10] A. Dogan, F. Özgüner, Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 308–323.
[11] W.C. Feng, T. Scogland, The green500 list: year one, in: IEEE International Symposium on Parallel & Distributed Processing (IPDPS 2009), 2009, pp. 1–7.
[12] M.R. Garey, D.S. Johnson, Computers and Intractability, vol. 174, Freeman, New York, 1979.

[13] R. Ge, X. Feng, K.W. Cameron, Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters, in: Proc. of the 2005 ACM/IEEE Conference on Supercomputing, 2005, pp. 34–45.
[14] T. Hagras, J. Janeček, A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems, Parallel Comput. 31 (7) (2005) 653–670.
[15] W. Huang, S.K. Oh, W. Pedrycz, A fuzzy time-dependent project scheduling problem, Inf. Sci. 246 (2013) 100–114.
[16] Intel, Intel Pentium m Processor Datasheet, 2004. <http://download.intel.com/support/processors/mobile/pm/sb/25261203.pdf>.
[17] T. Ishihara, S. Yamaguchi, Y. Ishitobi, T. Matsumura, Y. Kunitake, Y. Oyama, Y. Kaneda, M. Muroyama, T. Sato, Ample: an adaptive multi-performance processor for low-energy embedded applications, in: Symposium on Application Specific Processors (SASP 2008), 2008, pp. 83–88.
[18] V. Izosimov, P. Pop, P. Eles, Z. Peng, Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems, in: Proc. Design Automat. Test Europe Conf. (DATE'05), 2005, pp. 864–869.
[19] S. Kartik, C. Siva Ram Murthy, Task allocation algorithms for maximizing reliability of distributed computing systems, IEEE Trans. Comput. 46 (6) (1997) 719–724.
[20] M.A. Khan, Scheduling for heterogeneous systems using constrained critical paths, Parallel Comput. 38 (4) (2012) 175–193.
[21] K.H. Kim, R. Buyya, J. Kim, Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters, in: Proc. of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID'07), 2007, pp. 541–548.
[22] J. Kołodziej, S.U. Khan, Multi-level hierarchic genetic-based scheduling of independent jobs in dynamic heterogeneous grid environment, Inf. Sci. 214 (2012) 1–19.
[23] Y.C. Lee, A.Y. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, IEEE Trans. Parallel Distrib. Syst. 22 (8) (2011) 1374–1381.
[24] K. Li, Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers, IEEE Trans. Comput. (2012) 1668–1681.
[25] R. Melhem, D. Mossé, E. Elnozahy, The interplay of power management and fault recovery in real-time systems, IEEE Trans. Comput. 53 (2) (2004) 217–231.
[26] R. Min, T. Furrer, A. Chandrakasan, Dynamic voltage scaling techniques for distributed microsensor networks, in: Proc. of IEEE Computer Society Workshop on VLSI, 2000, pp. 43–46.
[27] G.C. Sih, E.A. Lee, A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures, IEEE Trans. Parallel Distrib. Syst. 4 (2) (1993) 175–187.
[28] R. Sridharan, N. Gupta, R. Mahapatra, Feedback-controlled reliability-aware power management for real-time embedded systems, in: Proc. Design Automat. Conf., 2008, pp. 185–190.
[29] X. Tang, K. Li, R. Li, B. Veeravalli, Reliability-aware scheduling strategy for heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 70 (9) (2010) 941–952.
[30] X. Tang, K. Li, M. Qiu, E.H.M. Sha, A hierarchical reliability-driven scheduling algorithm in grid systems, J. Parallel Distrib. Comput. 72 (4) (2012) 525–535.
[31] Y. Tian, J. Boangoat, E. Ekici, F. Özgüner, Real-time task mapping and scheduling for collaborative in-network processing in DVS-enabled wireless sensor networks, in: 20th International Parallel and Distributed Processing Symposium (IPDPS'06), 2006, pp. 10–19.
[32] H. Topcuoglu, S. Hariri, M.y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.
[33] V. Venkatachalam, M. Franz, Power reduction techniques for microprocessor systems, ACM Comput. Surv. (CSUR) 37 (3) (2005) 195–237.
[34] L. Wang, S.U. Khan, D. Chen, J. Kołodziej, R. Ranjan, C.Z. Xu, A. Zomaya, Energy-aware parallel task scheduling in a cluster, Future Gener. Comput. Syst. 29 (7) (2013) 1661–1670.
[35] M. Weiser, B. Welch, A. Demers, S. Shenker, Scheduling for reduced cpu energy, in: Mobile Computing, Springer, 1996, pp. 449–471.
[36] Wikipedia, Moore's law, 2012. <http://en.wikipedia.org/wiki/Moore's_law>.
[37] Wikipedia, Big data, 2014a. <http://en.wikipedia.org/wiki/Big_data>.
[38] Wikipedia, Tmall.com, 2014b. <http://www.tmall.com/>.
[39] Y. Xu, K. Li, J. Hu, K. Li, A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues, Inf. Sci. 270 (2014) 255–287.
[40] Y. Zhang, K. Chakrabarty, Energy-aware adaptive checkpointing in embedded real-time systems, in: Proc. Design Automat. Test Europe Conf., 2003, pp. 918–923.
[41] B. Zhao, H. Aydin, D. Zhu, On maximizing reliability of real-time embedded applications under hard energy constraint, IEEE Trans. Industr. Inf. 6 (3) (2010) 316–328.
[42] X. Zhong, C.Z. Xu, Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee, IEEE Trans. Comput. 56 (3) (2007) 358–372.
[43] D. Zhu, R. Melhem, B.R. Childers, Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems, IEEE Trans. Parallel Distrib. Syst. 14 (7) (2003) 686–700.
[44] D. Zhu, R. Melhem, D. Mossé, The effects of energy management on reliability in real-time embedded systems, in: IEEE/ACM International Conference on Computer Aided Design (ICCAD'04), 2004, pp. 35–40.