Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 370917, 15 pages
http://dx.doi.org/10.1155/2014/370917

Research Article

An Efficient Biobjective Heuristic for Scheduling Workflows on Heterogeneous DVS-Enabled Processors

Pengji Zhou and Wei Zheng

School of Information Science and Technology, Xiamen University, Xiamen 361005, China

Correspondence should be addressed to Wei Zheng; zhengw@xmu.edu.cn

Received 5 December 2013; Accepted 12 June 2014; Published 8 July 2014

Academic Editor: Aderemi Oluyinka Adewumi

Energy consumption has recently become a major concern to multiprocessor computing systems, of which the primary performance goal has traditionally been reducing execution time of applications. In the context of scheduling, there have been increasing research interests on algorithms using dynamic voltage scaling (DVS), which allows processors to operate at lower voltage supply levels at the expense of sacrificing processing speed, to acquire a satisfactory trade-off between quality of schedule and energy consumption. The problem considered in this paper is to find a schedule for a workflow, which is normally a precedence constrained application, on a bounded number of heterogeneous DVS-enabled processors, so as to minimize both makespan (overall execution time of the application) and energy consumption. A fast and efficient heuristic is proposed and evaluated using simulation with two real-world applications as well as randomly generated ones.

1. Introduction

During the last few decades, explosions in the volume of computation and/or data have stimulated a variety of researches on multiprocessor platforms (such as grids and clouds) to host complicated applications such as workflows [1, 2], which are widely used in the engineering, business, and science fields. It is not difficult to imagine that these powerful platforms, with a large (and still increasing) group of computing, storage, and connection equipment, must consume an enormous amount of energy. It has been estimated that the annual data center energy consumption in 2011 in the United States is over 100 billion kWh and at a cost of $7.4 billion [3]. According to [4], in the United States, energy consumed by the information and communication technology equipment is roughly 8% of the total and will increase 50% within a decade. This, undoubtedly, will further deteriorate the environment with increasing CO_2 emission.

The increasingly challenging energy problem urges growing need in developing energy-efficient solutions for multiprocessor platforms. However, most of the current researches on resource management of these platforms (e.g., Condor [5], Pegasus [6], etc.) mainly focus on achieving performance goals like high performance, high throughput, high reliability, and/or high availability to cater to users' requirements. As a result, most existing multiprocessor platforms generally lack capability on energy saving. This renders energy consumption problem an urgent and crucial issue to address.

Recent advancement in hardware technologies [7] (including dynamic voltage and frequency scaling, resource hibernation, memory optimization, solid state drives, energy-efficient computer monitors, etc.) have dealt with the energy consumption issues to some extent. However, it still remains a serious concern for software techniques such as scheduling algorithms (especially in a multiprocessor platform) to achieve substantial energy saving.

In this paper we consider workflow scheduling based on DVS, as it has demonstrated to be a promising technique in an abundance of literatures [8–12]. DVS enables processors to dynamically adjust voltage supply levels (VSLs) and CPU frequencies aiming to reduce power consumption, while an acceptable amount of performance sacrifice is paid as the expense.

With the aim at simultaneously minimizing makespan and energy consumption, the general form of the problem we considered here boils down to biobjective DAG scheduling,

as we assume every workflow application is represented by a directed acyclic graph (DAG). In particular, we focus on DAG scheduling for admission control of service- and market-oriented computing environments such as clouds, where a user and a service provider need to reach an agreement before the execution of the user application, and users are free to choose among different service providers. In such a scenario, a service provider needs the DAG scheduling return a competitive makespan (to attract customers) and a low energy consumption (for energy saving). Moreover, the scheduling should be performed in *short time* as users normally require a real-time response. There have been a few biobjective DAG scheduling heuristics in the literature. Some of these heuristics may provide quick response, but their performance leaves a considerable space to improve. Other heuristics may exhibit satisfactory performance but the scheduling cost is extremely high, and therefore not particularly suitable for the scenario discussed above. The need for fast and efficient DAG scheduling heuristics, suitable for real admission control of clouds, motivates the work presented in this paper.

This paper presents a new biobjective heuristic with the objective to simultaneously provide effective DVS-based DAG scheduling and fast scheduling time. Our heuristic is an enhancement of energy conscious scheduling heuristic (ECS) [11], which could make a quick scheduling decision, whereas the scheduling performance is often limited due to local optimum. With deliberation, we refine the core of ECS, namely, propose a novel objective function used by the RS (relative superiority) and a new criteria used by the MCER (makespan-conservative energy reduction technique) phases of ECS, to derive a new heuristic. The comparison results obtained from our extensive evaluation show that our approach can make significant improvement on both makespan optimization and energy reduction while still meeting real-time response requirement. This indicates that our approach can be easily applied to admission control of service- and market-oriented computing systems.

The remainder of the paper is organized as follows. Section 2 describes the background and related work. Section 3 describes the models used in our study and specifies the problem to be addressed. The proposed scheduling approach is presented in Section 4 with an illustrative example. The results of our comparative evaluation are shown in Section 5. Finally, the paper is concluded in Section 6.

2. Related Work

Dozens of static DAG scheduling heuristics aiming at minimizing makespan for heterogeneous multiprocessor systems have been presented in the literature. These heuristics are designed following different design principles. We hereby roughly classify these heuristics into list-scheduling algorithms [13–16], duplication-based algorithms [17–19], clustering algorithms [20, 21], and guided random search methods [22, 23]. Apparently, all these heuristics are different with our study in that their scheduling does not take energy consumption into account.

As DVS is a promising energy saving technique that can be incorporated into scheduling, a large number of scheduling algorithms based on DVS have been proposed for diverse applications and computing platforms. The majority of these DVS-based scheduling heuristics are conducted on homogeneous computing systems [9, 10, 24, 25], or single-processor systems [3, 26, 27], or focused on independent tasks [28–30]. These heuristics cannot address issues like task dependency and processor heterogeneity, which are addressed in our study.

There are also DVS-based scheduling heuristics focusing on DAG applications as well as heterogeneous systems. Huang et al. [12] proposed an enhanced energy-efficient scheduling algorithm to reduce energy consumption while meeting performance-based service level agreement (e.g., deadline constraint). This algorithm exploited the slack room between initially scheduled tasks and reallocated them in a global manner to achieve power saving. Unlike this work, applications considered in our study are not deadline-constrained, and the evaluation of the quality of schedules should be measured on both makespan and energy consumption.

Evolutionary techniques (i.e., genetic algorithm) have been widely applied to various problems (i.e., energy supply [31], space allocation [32], and multiobjective scheduling [33], etc.). Mezmaz et al. [34] proposed a hybrid genetic algorithm using DVS to simultaneously minimize makespan and energy consumption. Algorithms based on evolutionary techniques normally perform well on optimization. However, these algorithms usually require significantly high scheduling costs, even though modification may be applied to improve their efficiency [35]. As a result, these algorithms are naturally too time-consuming for admission control of clouds where a real-time response is required.

Energy-conscious scheduling heuristic (ECS) [11] is a list-scheduling algorithm aiming at simultaneously minimizing makespan and energy consumption with a low complexity. The heuristic consists of two phases. In the first phase, the heuristic applies bottom-level ranking to prioritize tasks, and then, in turn, selects the processor and the VSL for the current task so that the devised objective function, which is defined as relative superiority (RS), can be maximized. After the first phase, a temporary schedule is generated. In the second phase, a new criterion is used, which is defined as makespan-conservative energy reduction technique (MCER). That is, for each prioritized task in the current schedule, all of other combinations of task, processor, and VSL are checked to see whether any of these combinations reduces the energy consumption of the task without increasing the current makespan. If so, such a combination is applied to obtain a new schedule. After the second phase, the newest schedule is returned as the scheduling result. Evaluation results demonstrate that ECS significantly outperforms energy unconscious heuristics on energy consumption. However, the RS and MCER used by ECS, which are the cores of the algorithm, consider only local optimality. As a result, the scheduling decisions made by ECS tend to be confined to a local optimum. This motivates our work to propose novel objective function and criteria and devise a new heuristic.

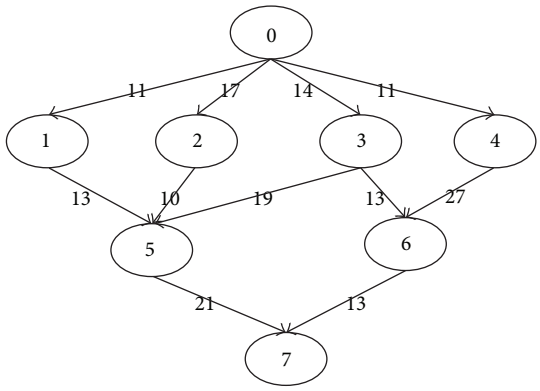


Figure 1: A simple DAG G.

The experimental results presented in Section 5 clearly show that our approach obtains schedules which are better than those found by ECS on both makespan optimization and energy reduction.

3. Problem Description

In this section, we describe the application model, the system model, and the energy model that used in our work and then specify the problem we are going to address.

3.1. Application Model. We use a directed acyclic graph (DAG) to represent an application to be scheduled (shown in Figure 1 with its details in Table 3). In a DAG, nodes denote tasks and edges that represent data transmission between tasks. In our work, we use $G = (N, E)$ to represent a DAG, which consists of a set of nodes N and a set of edges E. A node $i \in N$ represents the corresponded task and an edge $(i, j) \in E$ represents the intercommunication and precedence constraint between node i and j. For an edge (i, j), i is called a parent node of j, and j is called a child node of i. A child node cannot start execution until all of its parents have finished and all the required data transmission has arrived. Parentless nodes are called *source nodes*; childless node are called *sink node*. Apparently, an entry node of G must be a source node and an exit node a sink node. For standardization, we specify in this paper that a DAG has only a single entry node and a single exit node. One can easily see that all DAGs with multiple entry or exit nodes can be equivalently transformed to this standardization [36]. For illustration, a simple example DAG is shown in Figure 1, where the weight attached to each edge denotes the amount of data to be transmitted.

In order to meet precedence constraint, the start time and the finish time of task j on processor $q \in N$ are computed by

$$\begin{aligned} \mathrm{ST}(j,q) &= \max\left\{\mathrm{FT}(l^*,q), \max_{k\in \mathrm{Par}_j}\{\mathrm{FT}(k,p_k) \right. \\ &\qquad \left. + \mathrm{TC}((k,p_k),(j,q))\}\right\}, \\ \mathrm{FT}(j,q) &= \mathrm{ST}(j,q) + \mathrm{EC}(j,q), \end{aligned} \tag{1}$$

where $\mathrm{EC}(j, q)$ represents the execution time of task j on processor q; $\mathrm{FT}(l^*, q)$ denotes the finish time of task l^* which is the currently last task on processor q; Par_j represents the set of all parent tasks of task j; p_k denotes the processor which task k is assigned to, and if there is no task assigned to processor q, $\mathrm{FT}(l^*, q)$ is equal to zero. In the case of the entry task, we hav

$$\mathrm{ST}\left(\mathrm{entrynode}, p_{\mathrm{entrynode}}\right) = 0. \tag{2}$$

After the scheduling is completed, the *makespan* of the schedule, is defined as

$$\mathrm{makespan} = \max_{k\in N}\mathrm{FT}\left(k, p_k\right). \tag{3}$$

3.2. System Model. We consider a set of DVS-enabled heterogeneous processors which are fully interconnected and equally capable of running any applications. All the processors can run at different voltage and frequency levels. While the processor is in idle, it stays at its lowest voltage and lowest frequency level for the maximal energy saving [37]. Hereby we assume a set of DVS-enabled processors (denoted by P) that are fully connected. It is assumed that the time needed to transmit per unit of data from one processor to another, named *transmission rate*, is constant and preknown (as illustrated in Table 2). Therefore, the time needed to transmit data from one processor to another, named *transmission latency*, is computed by

$$\mathrm{TC}\left((i,p),(j,q)\right) = \mathrm{TD}(i,j) \times \mathrm{TR}(p,q), \tag{4}$$

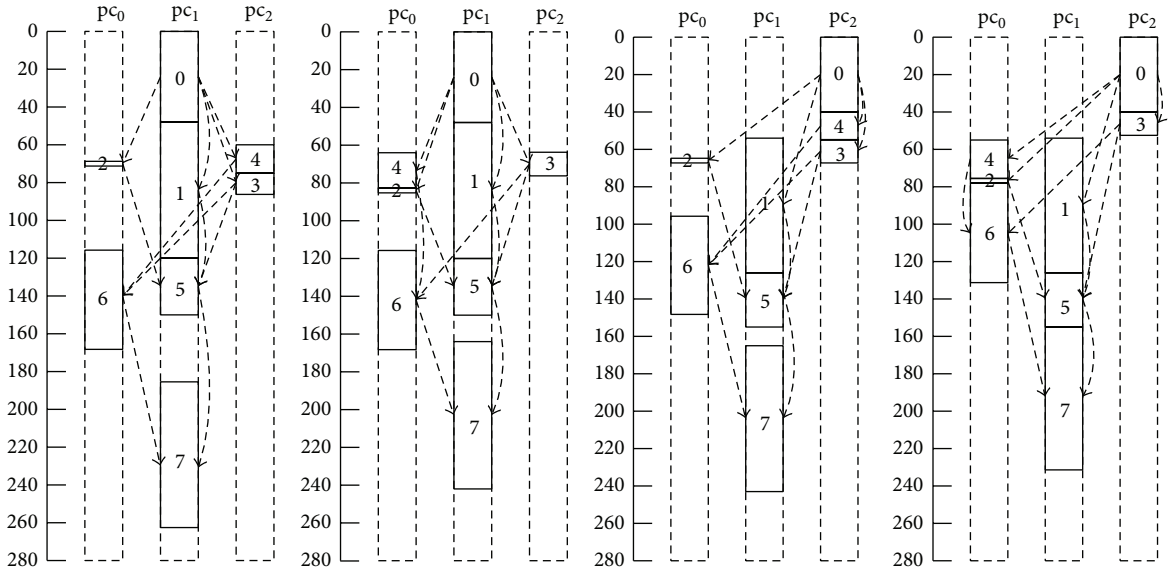
where $\mathrm{TD}(i, j)$ denotes the amount of transmitted data from task i to j and $\mathrm{TR}(p, q)$ if task i and j are allocated to the same processor, the transmission latency is zero. It is also assumed that one processor can only run one task at a time and no preemption is considered.

Each processor can operate in a set of voltage supply levels (VSL, denoted by V), each of which is corresponded to a specific relative speed (as illustrated in Table 1). For task n_i, we assume its execution time on a processor p, which operates on VSL 0 (denoted by $\mathrm{EC}(i, p, v_{p,0})$), is preknown; thereby, the execution time of n_i on a different VSL j (denoted by $\mathrm{EC}(i, p, v_{p,j})$) can be obtained by the ratio of $\mathrm{EC}(i, p, v_{p,0})$ and the relative speed of VSL j.

3.3. Energy Consumption Model. We adopt the energy model used in [11], which is derived from the power consumption model in complementary metal-oxide semiconductor (CMOS) logic circuits. Since we assume the processors consume a certain amount of energy while idling, the total energy consumption of the execution for a DAG is comprised of direct and indirect energy consumption. The direct energy consumption is defined as

$$E_d = \sum_{i=1}^{n} \alpha V_i^2 \Delta t_i, \tag{5}$$

where n is the number of tasks, α is a device related constant, V_i is the voltage on which the processor operates



(a) Result of RS of ECS (b) Result of MCER of ECS (c) Result of PCS of EECS (d) Result of GES of EECS

Figure 2: Schedules of G with EECS and ECS algorithms.

Table 1: Voltage relative speed pairs.

Level	pc_0		pc_1		pc_2	
	Voltage (v_k)	Speed (%)	Voltage (v_k)	Speed (%)	Voltage (v_k)	Speed (%)
0	1.60	100	1.20	100	2.00	100
1	1.40	85	1.10	90	1.70	80
2	1.20	70	1.00	80	1.40	60
3	1.00	55	0.90	70	1.10	40
4	0.80	40	0.80	60		
5			0.70	50		

Table 2: Transmission rate between different processors.

Connected processors	Transmission rate
pc_0 and pc_1	1.27
pc_0 and pc_2	1.53
pc_1 and pc_2	1.10

Table 3: Computation cost with VSL 0.

Task	pc_0	pc_1	pc_2	Task	pc_0	pc_1	pc_2
0	36	24	16	4	8	35	6
1	27	36	41	5	25	15	36
2	1	33	48	6	21	21	33
3	30	6	5	7	34	39	31

when executing task i, and Δt_i is the amount of time taken for n_i's execution. On the other hand, the indirect energy consumption is defined as

$$E_i = \sum_{j=1}^{p} \sum_{d_{j,k} \in D_j} \alpha V_{j,\text{low}}^2 \Delta w_{j,k}, \tag{6}$$

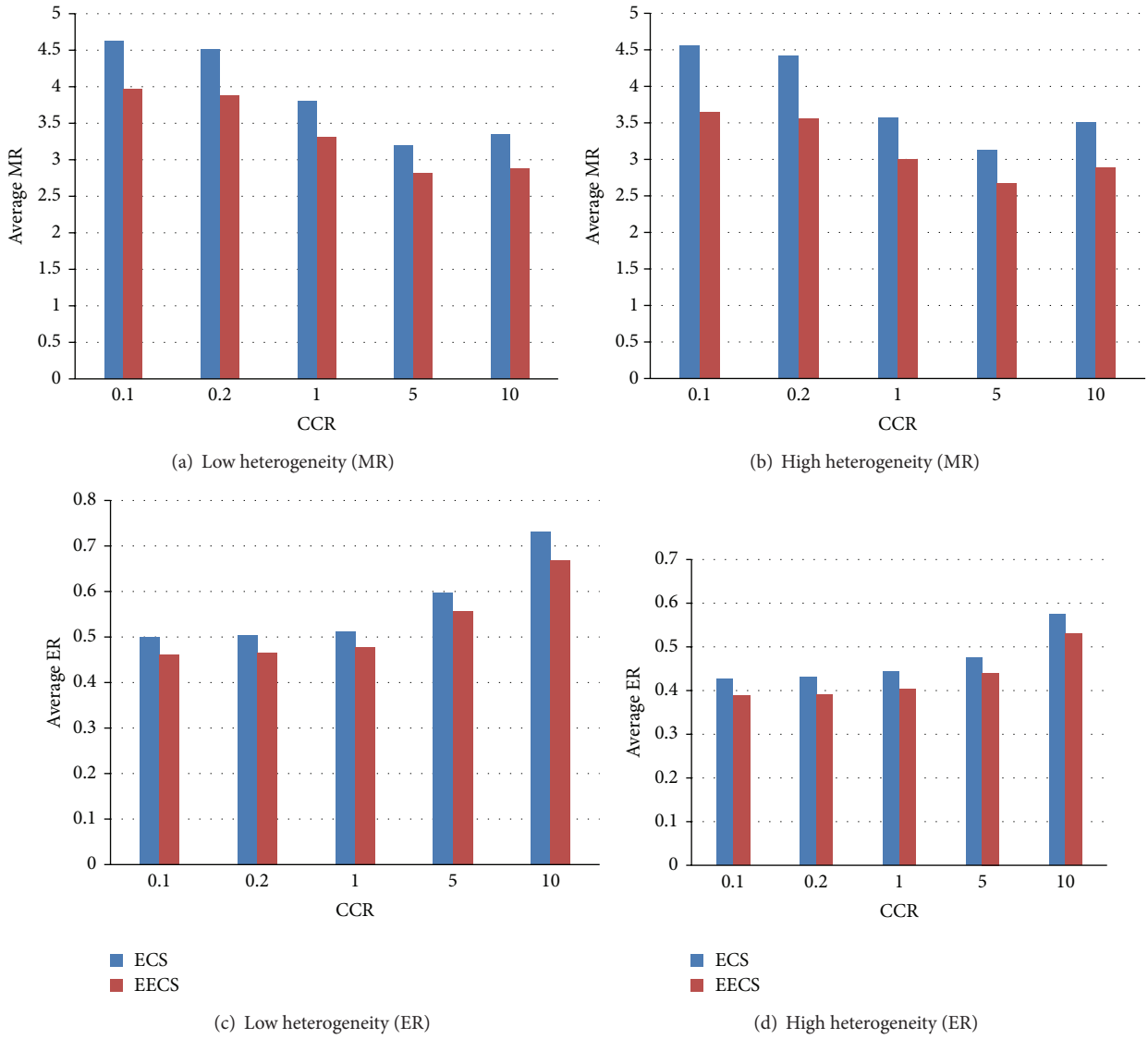
where p is the number of processors, D_j is the set of idling slots (between time 0 and the makespan) on processor p_j, $V_{j,\text{low}}$ is the lowest supply voltage on p_j, and $\Delta w_{j,k}$ is the amount of idling time for $d_{j,k}$. Then, the total energy consumption is defined as

$$E_{\text{total}} = E_d + E_i. \tag{7}$$

3.4. Scheduling Problem. The scheduling problem in this study is allocating n tasks in a DAG to p DVS-enabled heterogeneous processors, to simultaneously minimize makespan and energy consumption while still meeting precedence constraints between tasks. We assume all DAGs start execution at time 0 and the makespan is defined as the latest finish time of n tasks after the scheduling is completed.

4. Methodology

In this section, we present the proposed new heuristic enhanced energy conscious scheduling heuristic (EECS), as well as a simple example for illustration purpose.



(a) Low heterogeneity (MR)

(b) High heterogeneity (MR)

(c) Low heterogeneity (ER)

(d) High heterogeneity (ER)

Figure 3: Laplace, 49 nodes (different heterogeneities of processors).

Table 4: Task priorities of G.

n_i	0	1	2	3	4	5	6	7
b-level	178.50	138.87	127.63	125.67	128.00	87.30	76.57	34.67

Table 5: The schedule results of EECS and ECS for G.

	Result of RS of ECS	Result of MCER of ECS	Result of PCS of EECS	Result of GES of EECS
Makespan	264.42	242.8	243.32	232.1
Energy	25946.87	23915.8	23967.02	22861.85

Table 6: Experimental parameters.

Parameter	Value
The number of tasks	$U(20, 200)$
CCR	{0.1, 0.2, 1, 5, 10}
The number of processors	{3, 5, 8}
Processors heterogeneity	{low, high}

4.1. Proposed Heuristic. As presented in Algorithm 1, our heuristic first prioritizes tasks based on bottom-level ranking (denoted by *b-level*), which is computed by adding the average computation and communication costs along the longest path of the exit node in the DAG. Next, Algorithm 2 is applied to the prioritized tasks to generate an initial schedule. However, scheduling decisions made in Algorithm 2 are

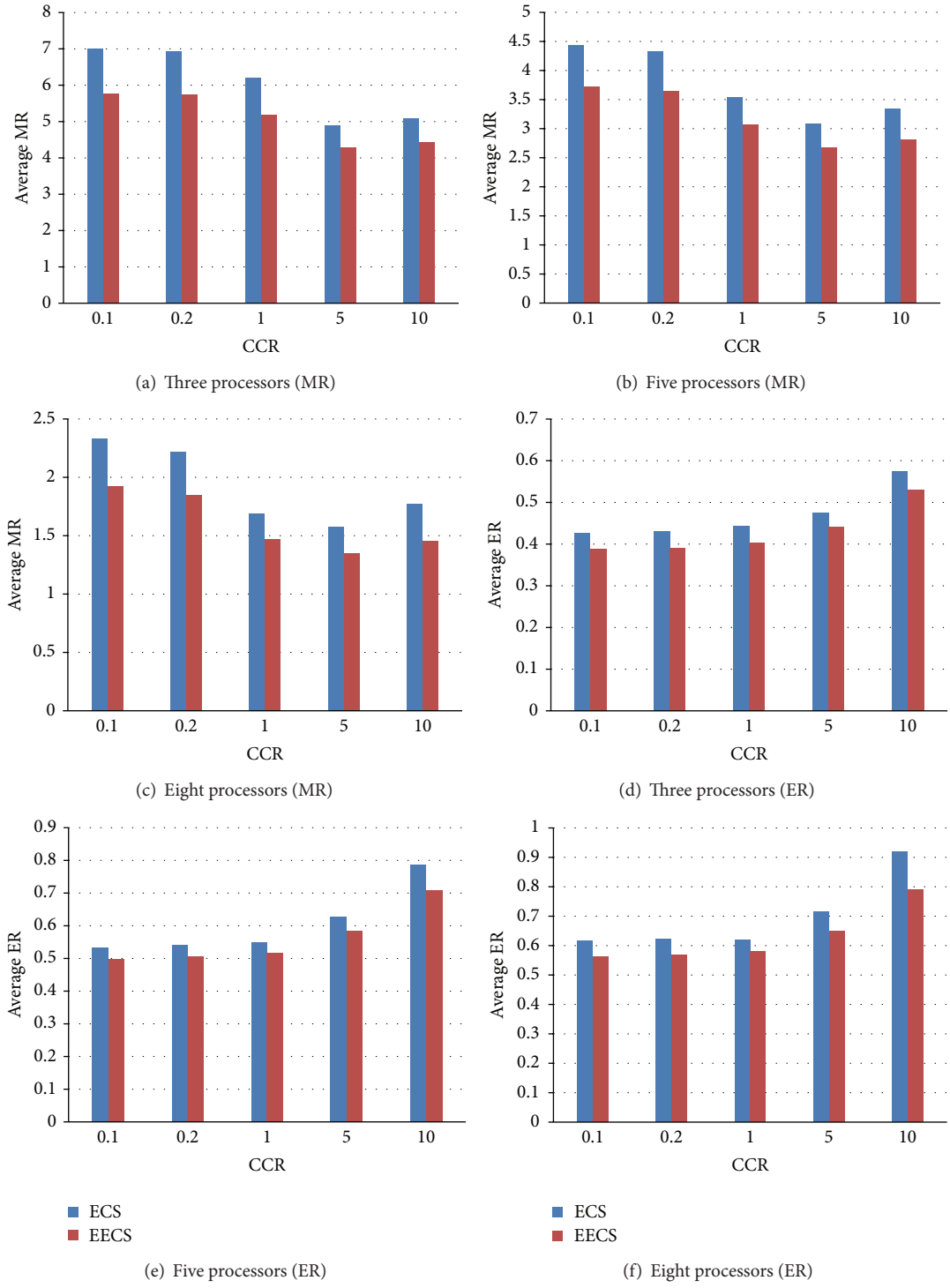
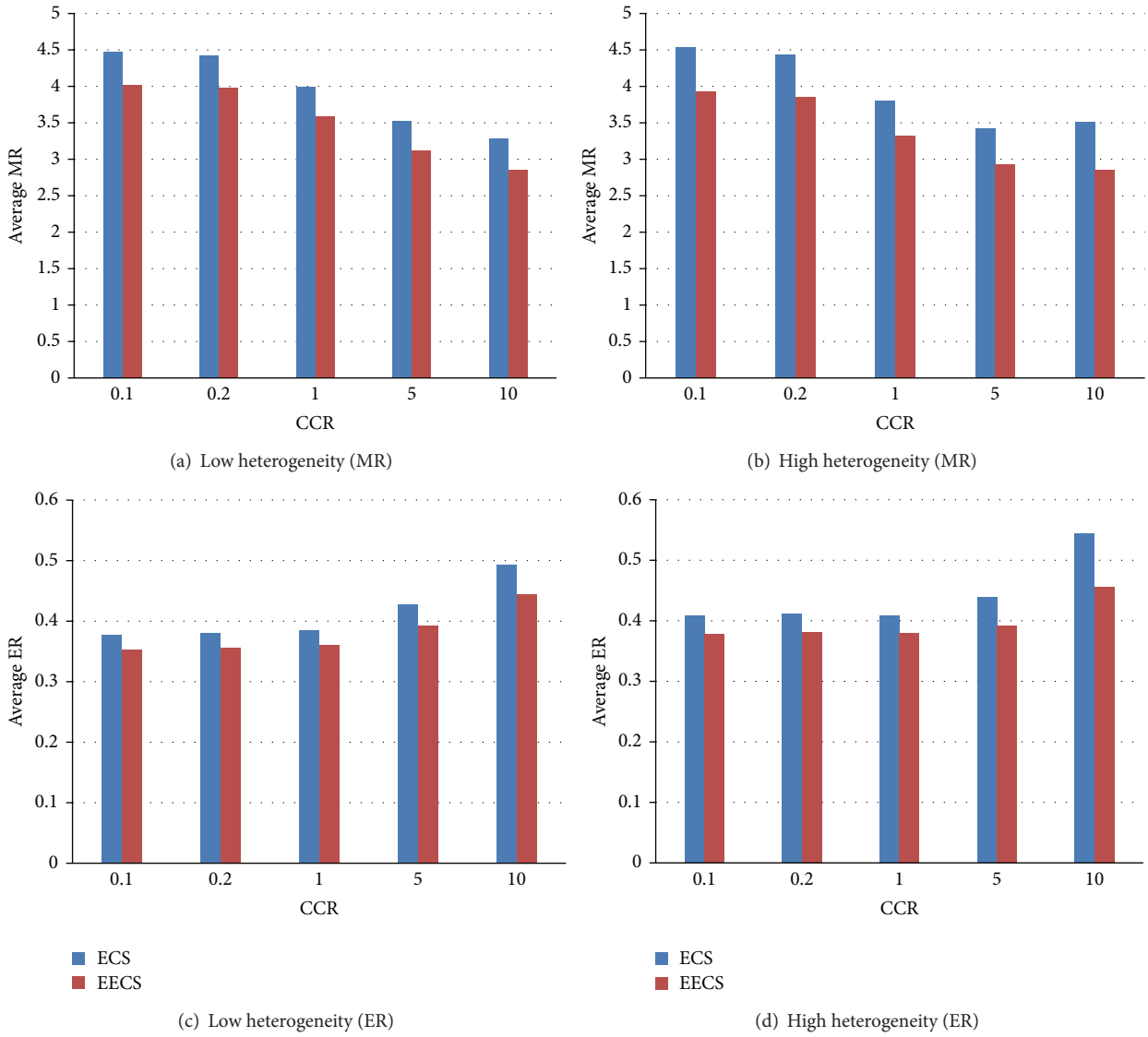


Figure 4: Laplace, 49 nodes (different sizes of processors).

inevitably limited by local greed. Therefore, the generated schedule is adjusted by Algorithm 3 for further optimization.

Algorithm 2 explains how the scheduling decision is made for each task in the initial schedule. We make scheduling decisions for tasks in turn. In each turn, one task is assigned a specific processor with a specific VSL, which is picked up from all possible combinations of processor and VSL, for optimum. Note that our scheduling aims at minimization on two objectives (i.e., makespan and energy consumption), which normally conflict with each other. This indicates the evaluation of a processor-VSL combination is not straightforward. In order to make a comparison between



(a) Low heterogeneity (MR)

(b) High heterogeneity (MR)

(c) Low heterogeneity (ER)

(d) High heterogeneity (ER)

Figure 5: LIGO, 77 nodes (different heterogeneities of processors).

Table 7: Comparative results of different sizes of processors.

	Improvement by EECS over ECS					
	3		5		8	
	Makespan	Energy	Makespan	Energy	Makespan	Energy
Random	10.2%	6.1%	11.01%	7.52%	15.12%	11.81%
LIGO	10.54%	7.25%	12.18%	8.01%	14.28%	9.67%
Laplace	15.73%	8.34%	14.98%	7.49%	16.26%	9.88%
Average	12.16%	7.23%	12.73%	7.67%	15.22%	10.45%

two combinations, we devise substitution score (SUBS). For task n_i, SUBS(n_i, p', v', p, v) quantifies the score gained if a processor-VSL combination (p, v) is replaced by (p', v'). SUBS deliberately takes into account the trade-off between makespan minimization and energy reduction. As defined in (8), SUBS is a sum of three factors. The first factor is local energy factor, which is the difference of energy caused by the substitution with normalization by the energy consumption of current task. The second factor is local execution time factor, which is the difference of task execution time caused by the substitution with normalization by the execution time of current task. The third factor is makespan factor, which is the difference of task finish time caused by the substitution with normalization by the execution time of current task.

Input: A DAG $G(N, E)$ and a set P of DVS-enabled processors.
Output: A schedule S of G onto P.
(1) Compute the weights of nodes and edges averaged over differen processors.
(2) Compute the bottom-level ranking for each node.
(3) Sort all tasks in the no-ascending order of bottom-level and put them into list L.
(4) Generate an initial schedule with consideration on makespan-energy tradeoff (by Algorithm 2).
(5) Adjust the schedule for global energy saving. (by Algorithm 3).

Algorithm 1: EECS heuristic outline.

(1) **for** each sorted task n_i in L **do**
(2) let p_{opt} be p_0.
(3) let v_{opt} be $v_{p_0,0}$.
(4) **for** each processor p_j in P **do**
(5) **for** each voltage $v_{p_j,k}$ in V **do**
(6) Compute $S_{\text{stay}} = \text{SUBS}(n_i, p_{\text{opt}}, v_{\text{opt}}, p_j, v_{p_j,k})$ as defined in (8).
(7) Compute $S_{\text{change}} = \text{SUBS}(n_i, p_j, v_{p_j,k}, p_{\text{opt}}, v_{\text{opt}})$ as defined in (8).
(8) **if** S_{stay} is greater than S_{change} **then**
(9) Assign p_j and $v_{p_j,k}$ to p_{opt} and v_{opt}, respectively.
(10) **end if**
(11) **end for**
(12) **end for**
(13) Allocate task n_i on p_{opt} with v_{opt}.
(14) **end for**

Algorithm 2: Pairwise comparison and selection.

As defined in (8), in the case of $p = p'$, the makespan factor is ignored, as the sign of makespan factor is always in accordance with the sign of local execution time factor:

$$\text{SUBS}\left(n_i, p', v', p, v\right) = \begin{cases} \dfrac{E_d\left(n_i, p', v'\right) - E_d\left(n_i, p, v\right)}{E_d\left(n_i, p, v\right)} + \dfrac{\text{EC}\left(n_i, p', v'\right) - \text{EC}\left(n_i, p, v\right)}{\text{EC}\left(n_i, p, v\right)} + \dfrac{\text{FT}\left(n_i, p', v'\right) - \text{FT}\left(n_i, p, v\right)}{\text{EC}\left(n_i, p, v\right)} & \text{if } p!=p' \\ \dfrac{E_d\left(n_i, p', v'\right) - E_d\left(n_i, p, v\right)}{E_d\left(n_i, p, v\right)} + \dfrac{\text{EC}\left(n_i, p', v'\right) - \text{EC}\left(n_i, p, v\right)}{\text{EC}\left(n_i, p, v\right)} & \text{otherwise,} \end{cases} \quad (8)$$

where, for task n_i on processor p with VSL v, $E_d(n_i, p, v)$ denotes the directed energy consumption of n_i, $\text{EC}(n_i, p, v)$ the execution time of n_i, and $\text{FT}(n_i, p, v)$ the finish time of n_i.

In Algorithm 3, for each scheduled task, we check whether there exists another processor-VSL combination, which, by replacing the currently scheduled combination, can reduce the total energy consumption (different with the MCER technique used in ECS, which consider only the energy consumption of the current task) without increasing the makespan. If so, the replacement will be enforced.

Table 8: Comparative results of different heterogeneities of processors.

	Improvement by EECS over ECS			
	Low		High	
	Makespan	Energy	Makespan	Energy
Random	11.0%	7.25%	14.14%	9.49%
LIGO	10.83%	7.66%	14.42%	10.1%
Laplace	13.51%	7.52%	17.84%	9.75%
Average	11.78%	7.48%	15.47%	9.78%

Based on the above description, it is not difficult to compute that the complexity of our heuristic is $O(n \log n + 2((e + n)pv))$, where n is the number of DAG nodes, e the number of DAG edges, p the number of processors, and v the number of VSLs.

4.2. An Example. A simple DAG with 8 nodes is used here for illustration purpose. Figure 1 shows the DAG structure and the size of data to transmit between two interdependent tasks. Three processors (as depicted in Table 1) are assumed to run the DAG, and the execution time of each task on each processor is provided in Table 3. Additionally, Table 2 provides the data transmission rates among these processors.

Table 4 provides the *b-level* results computed for each node of the DAG example. According to these results, the tasks are sorted as follows: {0, 1, 4, 2, 3, 5, 6, 7}.

(1) **for** each task n_i sorted in L **do**
(2) let p_{opt} be the processor on which n_i is currently scheduled.
(3) let v_{opt} be the VSL to which n_i is currently assigned.
(4) **for** each processor p_j in P **do**
(5) **for** each voltage $v_{p_j,k}$ in V **do**
(6) Tentatively reallocate n_i onto p_j with v_{p_j}.
(7) Recompute the makespan.
(8) Recompute the total energy consumption E_{total} as defined in (7).
(9) **if** no increase in makespan and the total energy consumption is reduced **then**
(10) Assign p_j, $v_{p_j,k}$ to p_{opt},v_{opt}, respectively.
(11) Update the makespan and the total energy consumption.
(12) **end if**
(13) **end for**
(14) **end for**
(15) Allocate n_i on p_{opt} with v_{opt}.
(16) **end for**

Algorithm 3: Global energy saving.

Table 9: Comparative results of Figure 5: LIGO, 77 nodes (different heterogeneities of processors).

	ECS				EECS			
	Low heterogeneity		High heterogeneity		Low heterogeneity		High heterogeneity	
	MR (%)	ER (%)	MR (%)	ER (%)	MR (%)	ER (%)	MR (%)	ER (%)
0.1	4.47	0.37	4.53	0.41	4.02	0.35	3.93	0.38
0.2	4.42	0.38	4.44	0.41	3.98	0.36	3.85	0.38
1	4.0	0.39	3.81	0.40	3.59	0.36	3.32	0.38
5	3.53	0.43	3.43	0.44	3.12	0.39	2.93	0.39
10	3.28	0.49	3.52	0.54	2.85	0.44	2.85	0.46

Figure 2(a) depicts the schedule generated by the first phase (i.e., RS) of ECS, and Figure 2(b) is the schedule finally obtained by ECS after applying MCER. Figures 2(c) and 2(d) show the schedules generated by the PCS phase and the GES phases of EECS, respectively. The corresponding makespan and energy consumption for each schedule is provided in Table 5.

In this specific example, the PCS phase of EECS generates a better schedule (with shorter makespan and less energy consumption) than the one obtained by the RS phase of ECS. By comparing Figures 2(c) and 2(d), we clearly see the effectiveness of GES on energy reduction without increasing the makespan. Although for ECS, MCER can also improve the schedule quality obtained by RS. However, the final schedule of ECS is still 4.41% down on makespan minimization and 4.60% down on energy reduction, in comparison with the result obtained by EECS. This implies that EECS can outperform ECS on both minimizing makespan and reducing energy. We verify this implication in the next section.

5. Performance Evaluation

In this section, we compare our algorithm (EECS) with ECS. We consider DAGs derived from real-world workflow applications and a simulated heterogeneous system, which consists of processors with DVS parameter setting derived from real CPU models. Simulation results demonstrate the significant improvement our algorithm makes both on makespan optimization and energy saving.

5.1. Experimental Setting. In our evaluation, we considered randomly generated DAGs and two real-world applications, which are LIGO [38] with 77 nodes and Laplace equation solver [39] with 49 nodes. When generating random DAGs, we followed the method presented in [40]. Figure 1 illustrates how a random DAG looks like. Note that the node number of LIGO and Laplace is fixed, while the node number of a random DAG randomly selected from the range of [20, 200].

We also considered different numbers of resources: 3, 5, and 8. All processors are DVS-enabled and the VSL parameter is randomly selected from Table 1. In order to model task execution times, we adopted the method presented in [41]. In this method, in brief, two values are selected from a uniform distribution in a certain interval. The product of the two selected values is computed and adopted as a generation of one task execution time. We classified the task execution times generated from the interval [10, 50] into low heterogeneity, those from [10, 1000] into high heterogeneity.

The computation and communication ratio (CCR) is a measure that indicates whether the DAG is communication intensive, computation intensive, or moderate. The definition

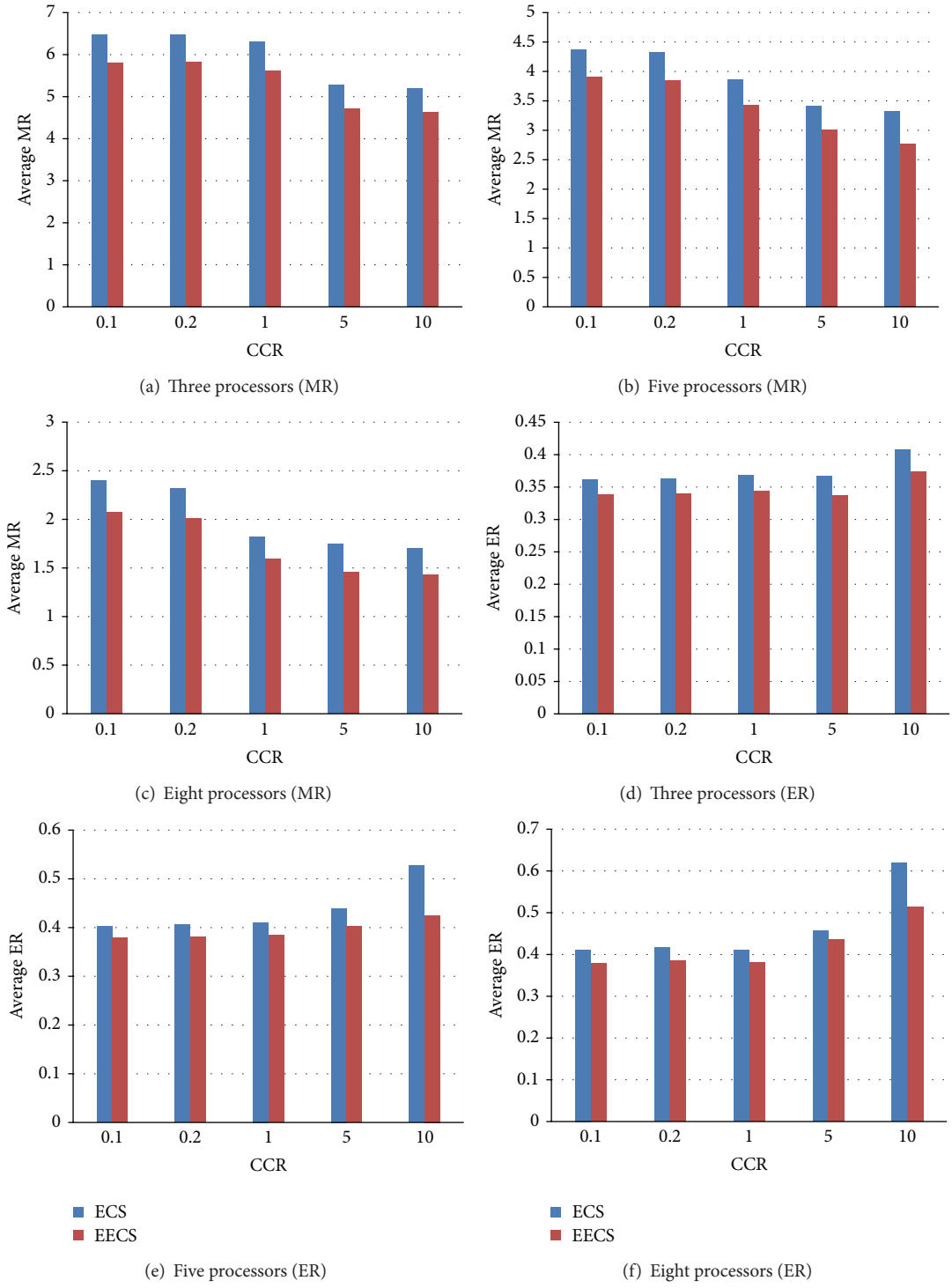


Figure 6: LIGO, 77 nodes (different sizes of processors).

of CCR is the ratio between the average communication cost and the average computation cost on the target system. We considered five specific CCR values: 0.1, 0.2, 1.0, 5, and 10. With a set of generated task execution times, the communication costs of the tasks were randomly generated to keep consistency with the given CCR.

For every competing heuristic (ECS and EECS), the number of experiments conducted is 45000. Table 6 summarized the parameters used in our experiments. Specifically, for each type of DAG, the base DAG set consists of 500 random samples. This figure is combined with 5 different CCRs, 3 different numbers of processors, 2 different types

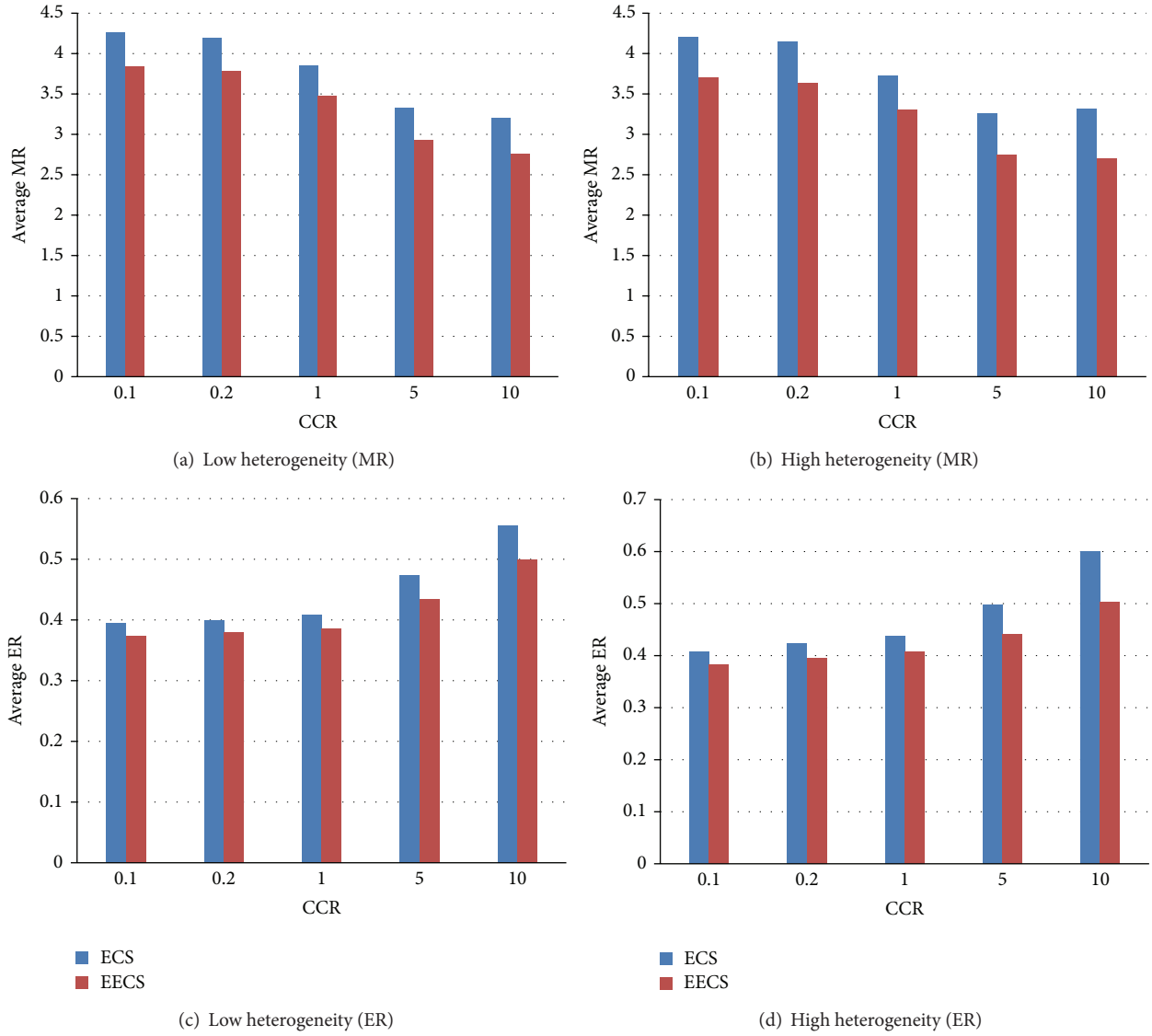


Figure 7: Random DAGs (different heterogeneities of processors).

of heterogeneity, and 3 different DAG types, which leads to the result of 45000. In each experiment, every algorithm is used to generate a schedule with makespan and energy consumption. Hence, the total number of experiments in our evaluation is 90000 (two algorithms were evaluated).

Finally, all the experiments were implemented by Java and run on a PC with AMD A6 CPU running at 2.20 GHz with 4 GB memory.

5.2. Comparison Metrics. In our evaluation, we consider makespan and energy consumption are equally performance metrics. For a given schedule, its makespan is normalized to a lower bound, which is the sum of the execution and communication costs of tasks along the critical path (denoted by M_{cp}), while its energy consumption is normalized to a upper bound, which is the total energy consumption of the schedule in which every task is scheduled so that the energy consumption is maximized (denoted by $E_{\max}$).

Specifically, for each experiment, the performance of each heuristic (ECS and EECS) is normalized to MR (makespan ratio) and ER (energy ratio) defined as follows:

$$\mathrm{MR} = \frac{M}{M_{cp}}, \qquad \mathrm{ER} = \frac{E}{E_{\max}}, \tag{9}$$

where M is the makespan of the schedule and E the energy consumption of the schedule.

5.3. Experimental Results. The results for each of the two different scheduling heuristics on the three different types of DAGs (note that for each DAG, results impacted by number of processors and results impacted by heterogeneity are both considered; this results in 6 pairs) are shown in Figures 3, 4, 5, 6, 7, and 8. Particularly, the actual comparative results of Figures 5 and 6 are shown in Tables 9 and 10, respectively. The results are normalized to MR and ER, respectively, as

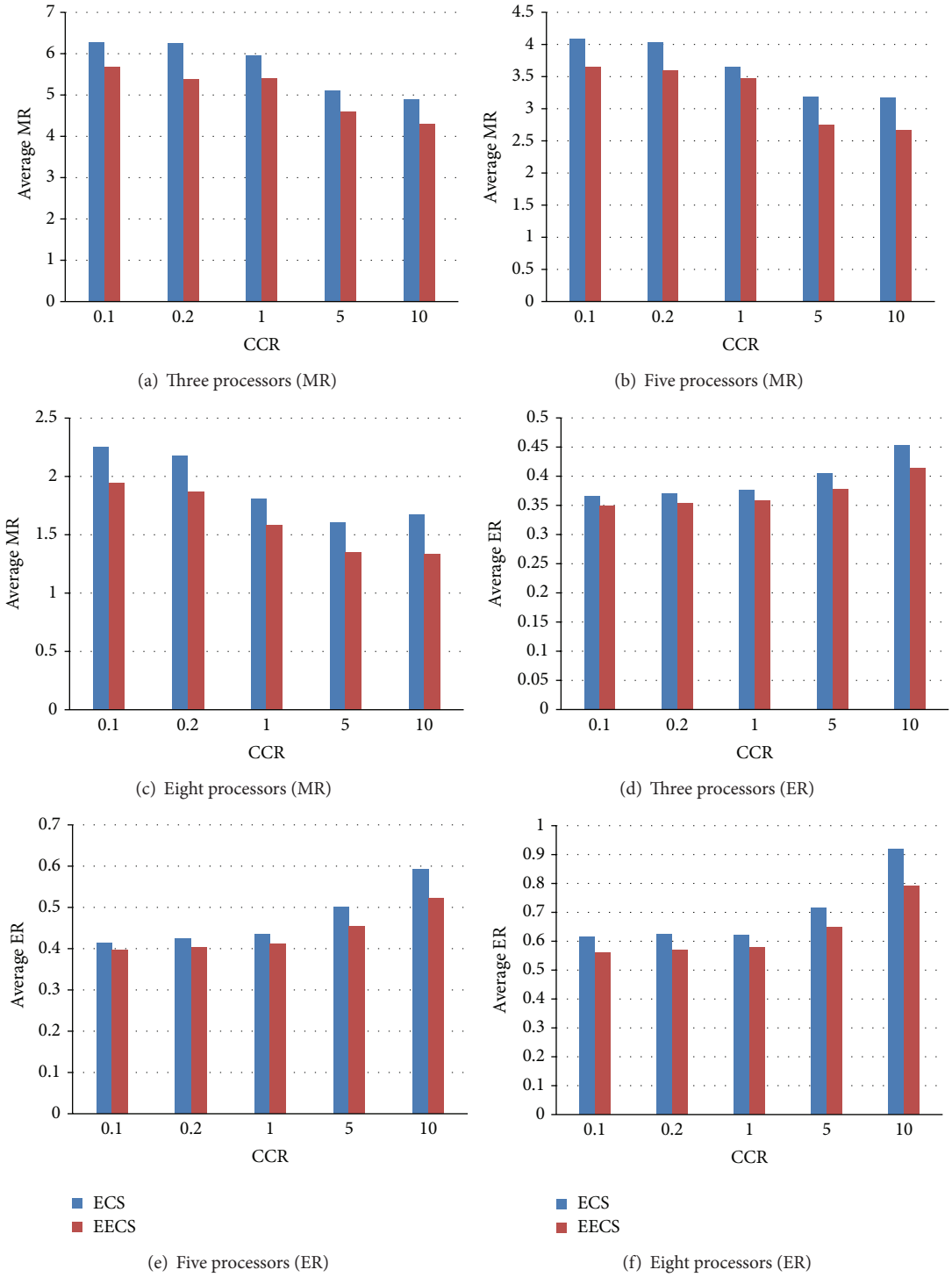


Figure 8: Random DAGs (different sizes of processors).

presented in Section 5.2. Recall that for each heuristic, all results are averaged over 500 runs.

In all cases depicted in the result figures, it is clear that EECS always obtained a MR and a ER less than their counterpart that ECS achieved. This indicates that EECS outperforms ECS in all cases on both makespan optimization and energy reduction.

It is interesting to see that the makespan improvement of EECS over ECS is somehow correlated with CCR. When Laplace is used, for both low and high heterogeneities, the MR difference between EECS and ECS is decreased, as CCR increases from 0.1 to 5. Then, this difference significantly increases, as CCR changes from 5 to 10. Such a variation of MR difference can also be observed when LIGO and random

Table 10: Comparative results of Figure 6: LIGO, 77 nodes (different sizes of processors).

	ECS						EECS					
	3		5		8		3		5		8	
	MR (%)	ER (%)	MR (%)	ER (%)	MR (%)	ER (%)	MR (%)	ER (%)	MR (%)	ER (%)	MR (%)	ER (%)
0.1	6.48	0.36	4.38	0.40	2.40	0.41	5.81	0.34	3.90	0.38	2.08	0.38
0.2	6.47	0.36	4.33	0.41	2.32	0.42	5.82	0.34	3.84	0.38	2.01	0.38
1	6.30	0.37	3.86	0.41	1.82	0.41	5.62	0.35	3.43	0.39	1.59	0.38
5	5.27	0.38	3.42	0.44	1.75	0.46	4.72	0.33	3.02	0.40	1.45	0.44
10	5.20	0.41	3.32	0.53	1.70	0.62	4.64	0.37	2.78	0.43	1.43	0.51

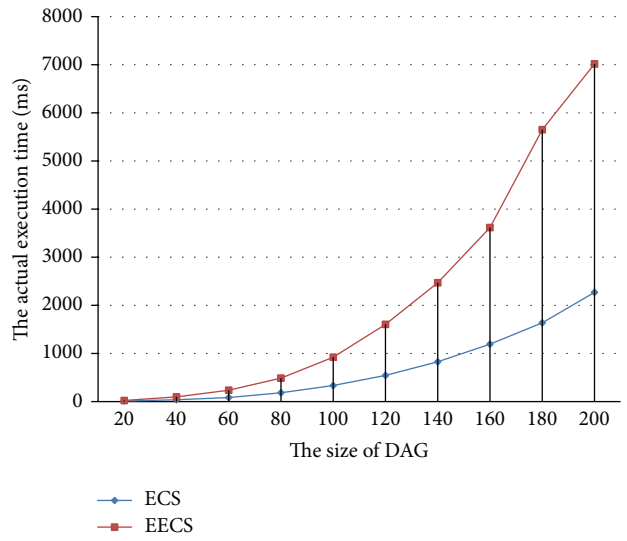


Figure 9: The actual execution time of compared heuristics.

is used with high heterogeneity. The ER difference between EECS and ECS varies little when CCR is not more than 5.0. However, a significant increase of ER difference can be seen when CCR changes from 5 to 10. These observations imply EECS may perform particularly better than ECS in the scenario where CCR is high.

Table 7 summarized the comparative results between ECS and EECS in terms of the change of number of processors. As the size of LIGO and Laplace is fixed, different settings of processor number may correspond to a specific scenario. Here, using 3 processors indicate a "resource-hungry" situation, 8 processors indicate a "resource-rich" situation, and 5 processors indicate a medium. When LIGO is used, the improvement of EECS over ECS, on both makespan and energy, increases as the number of processors grows. In the case of Laplace, such an improvement hits the lowest when 5 processors are used, while reaching the highest when 8 processors are used. So, it seems that EECS may obtain a greater improvement over ECS in a "resource-rich" scenario.

The comparative results between ECS and EECS in terms of different processor heterogeneities are summarized in Table 8. It is clearly suggested that the advantage of EECS over ECS may be magnified as the heterogeneity of processor turns from low to high.

From Tables 7 and 8, we can see that ECS may obtain a makespan up to 17.84% and energy consumption 10.1% more than EECS. Averagely, EECS significantly outperforms ECS by 12% on makespan minimization and 8% on energy reduction.

Aside from the comparison of scheduling performance, we assessed the running times of ECS and EECS for DAGs with different sizes. The results are shown in Figure 9. Although EECS and ECS are both based on list-scheduling, EECS needs a bit more running time than ECS as the computation involved in EECS is more complicated. However, as can be seen in the graph, when scheduling a DAG with 200 nodes, EECS only needs around 7 seconds on average. This suggests that EECS can still cope well with the real-time requirement of workflow scheduling for admission control of market-oriented systems.

6. Conclusion

This paper proposed EECS, a novel efficient biobjective DAG scheduling heuristic based on the enhancement to the energy conscious scheduling heuristic ECS. The proposed heuristic aims at simultaneously minimizing makespan and energy consumption with a low complexity. The experimental results suggest that EECS can significantly outperform the existing approach (i.e., ECS) on both makespan optimization and energy reduction. It also appears that EECS has a low execution time cost and thus is able to produce a schedule as a real-time response to users in market-oriented systems.

Based on the work in this paper, further work could try to examine the performance of EECS in an uncertain environment. Further study could investigate how EECS can cope with significant overestimation or underestimation of task execution time and assess its robustness against such uncertainties.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

The work is supported by National Natural Science Foundation of China (NSFC, Grant no. 61202361).

References

[1] E. Deelman, D. Gannon, M. S. Shields, and I. Taylor, "Workflows and e-Science: an overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.

[2] G. Juve, E. Deelman, G. B. Berriman, B. P. Berman, and P. Maechling, "An evaluat ion of the cost and performance of scientific workflows on Amazon EC2," *Journal of Grid Computing*, vol. 10, no. 1, pp. 5–21, 2012.

[3] R. Bianchini and R. Rajamony, "Power and energy management for server systems," *Computer*, vol. 37, no. 11, pp. 68–76, 2004.

[4] R. Brown, *Report to Congress on Server and Data Center Energy Efficiency: Public Law 109-431*, Lawrence Berk eley National Laboratory, 2008.

[5] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: a computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, 2002.

[6] K. Lee, N. W. Paton, R. Sakellariou, E. Deelman, A. A. A. Fernandes, and G. Mehta, "Adaptive workflow processing and execution in pegasus," *Concurrency Computation Practice and Experience*, vol. 21, no. 16, pp. 1965–1981, 2009.

[7] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Computing Surveys*, vol. 37, no. 3, pp. 195–237, 2005.

[8] K. Q. Li, "Energy efficient scheduling of parallel tasks on multiprocessor computers," *The Journal of Supercomputing*, vol. 60, no. 2, pp. 223–247, 2012.

[9] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters," in *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, pp. 541–548, May 2007.

[10] L. Wang, G. von Laszewski, J. Dayal, and F. Wang, "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS," in *Proceedings of the 10th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid '10)*, pp. 368–377, May 2010.

[11] 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*, vol. 22, no. 8, pp. 1374–1381, 2011.

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

[13] Y. Kwok and I. Ahmad, "Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506–521, 1996.

[14] R. Sakellariou and H. Zhao, "A hybrid heuristic for DAG scheduling on heterogeneous systems," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, pp. 111–124, Santa Fe, NM, USA, April 2004.

[15] G. C. Sih and E. A. Lee, "Compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 175–187, 1993.

[16] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[17] D. Bozdag, U. Catalyurek, and F. Ozguner, "A task duplication based bottom-up scheduling algorithm for heterogeneous environments," in *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS '06)*, 2006.

[18] S. Ranaweera and D. P. Agrawal, "A scalable task duplication based scheduling algorithm for heterogeneous systems," in *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, pp. 383–390, 2000.

[19] S. Ranaweera and D. P. Agrawal, "A task duplication based scheduling algorithm for heterogeneous systems," in *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, pp. 445–450, 2000.

[20] B. Cirou and E. Jeannot, "Triplet: a clustering scheduling algorithm for heterogeneous systems," in *Processings of the Internatioanl Conference on Parallel Processing Workshops*, pp. 231–236, 2001.

[21] J. C. Liou and M. A. Palis, "An efficient task clustering heuristic for scheduling DAGs on multiprocessors," in *Proceedings of the Workshop on Scheduling and Resource Management for Parallel and Distributed Processing*, pp. 152–156, 1996.

[22] T. D. Braun, H. J. Siegel, N. Beck et al., "A Comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems," in *Proceedings of the 8th Heterogeneous Computing Workshop (HCW '99)*, pp. 15–29, April 1999.

[23] M. Coli and P. Palazzari, "Real time pipelined system design through simulated annealing," *Journal of Systems Architecture*, vol. 42, no. 6-7, pp. 465–475, 1996.

[24] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters," in *Proceedings of the ACM/IEEE conference on Supercomputing*, p. 34, November 2005.

[25] D. Zhu, R. Melhem, and B. R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 7, pp. 686–700, 2003.

[26] S. S. Zhang and A. Chatha, "Approximation algorithm for the temperature-aware scheduling problem," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '07)*, pp. 281–288, November 2007.

[27] X. L. Zhong and C. Z. Xu, "Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee," *IEEE Transactons on Computers*, vol. 56, no. 3, pp. 358–372, 2007.

[28] Y. Yu and V. K. Prasanna, "Power-aware resoure allocation for independent tasks in heterogeneous real-time system," in *Proceedings of the 9th International Conference on Parallel and Distributed Systems*, pp. 341–348, 2002.

[29] X. Zhu and P. Lu, "A two-phase scheduling strategy for real-time applications with security requirements on heterogeneous clusters," *Computers and Electrical Engineering*, vol. 35, no. 6, pp. 980–993, 2009.

[30] X. Zhu, C. He, K. Li, and X. Qin, "Adaptive energy-efficient scheduling for real-time tasks on DVS-enabled heterogeneous clusters," *Journal of Parallel and Distributed Computing*, vol. 72, no. 6, pp. 751–763, 2012.

[31] W. F. Sacco, C. A. M. N. A. Pereira, P. P. M. Soares, and R. Schirru, "Genetic algorithms applied to turbine extraction optimization of a pressurized-water reactor," *Applied Energy*, vol. 73, no. 3-4, pp. 217–222, 2002.

[32] A. O. Adewumi and M. M. Ali, "A multi-level genetic algorithm for a multi-stage space allocation problem," *Mathematical and Computer Modelling*, vol. 51, no. 1-2, pp. 109–126, 2010.

[33] A. K. M. K. A. Talukder, M. Kirley, and R. Buyya, "Multiobjective differential evolution for scheduling workflow applications on global Grids," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 13, pp. 1742–1756, 2009.

[34] M. Mezmaz, N. Melab, Y. Kessaci et al., "A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems," *Journal of Parallel and Distributed Computing*, vol. 71, no. 11, pp. 1497–1508, 2011.

[35] M. M. Ali and A. Törn, "Population set-based global optimization algorithms: some modifications and numerical studies," *Computers and Operations Research*, vol. 31, no. 10, pp. 1703–1725, 2004.

[36] W. Zheng and R. Sakellariou, "Stochastic DAG scheduling using a Monte Carlo approach," *Journal of Parallel and Distributed Computing*, vol. 73, no. 12, pp. 1673–1689, 2013.

[37] R. Min, T. Furrer, and A. Chandrakasan, "Dynamic voltage scaling techniques for distributed microsensor networks," in *Proceedings of the IEEE Computer Society Workshop on Very Large Scale Integration*, pp. 92–99, 2000.

[38] D. Brown, P. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb, "A ca se study on the use of workflow technologies for scientific analysis: gravitational wave data analysis," in *Workflows for e-Science: Science Workflows for Grids*, pp. 39–59, 2006.

[39] M. Wu and D. D. Gajski, "Hypertool: a programming aid for message-passing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 330–343, 1990.

[40] W. Zheng, *Explorations in grid workflow scheduling [Ph.D. thesis]*, The University of Manchester, 2010.

[41] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen, "Task execution time modeling for heterogeneous computing systems," in *Proceedings of the 9th Heterogeneous Computing Workshop*, pp. 185–199, 2000.