

# A Multi-objective GRASP Algorithm for Joint Optimization of Energy Consumption and Schedule Length of Precedence-Constrained Applications

Johnatan E. Pecero\*, Pascal Bouvry\*, Hector J. Fraire Huacuja†, Samee U. Khan‡

*\*Computer Science and Communications*

*University of Luxembourg, L-1417 Luxembourg*

*Email: {firstname.lastname@uni.lu}*

*†Instituto Tecnológico de Cd. Madero, Mexico*

*Email: automatas2002@yahoo.com.mx*

*‡North Dakota State University, Fargo, ND 58108*

*Email: samee.khan@ndsu.edu*

**Abstract**—We address the problem of scheduling precedence-constrained scientific applications on a heterogeneous distributed processor system with the twin objectives of minimizing simultaneously energy consumption and schedule length. Previous research efforts on scheduling have focused on the minimization of a quality of service metric based on the completion time of applications (e.g., the schedule length). Recently, many researchers are working on the design of new scheduling algorithms that consider the minimization of energy consumption.

We report a new scheduling algorithm accounting for both objectives. The new scheduling algorithm is based on a multi-start randomized adaptive search technique (GRASP framework) that adopts Dynamic Voltage Scaling technique to minimize energy consumption. This technique enables processors to operate in different voltage supply levels at the cost of sacrificing clock frequencies. This multiple voltage implies a trade-off between the quality of the schedules and energy consumption. Therefore, the new proposed approach is designed as a multi-objective algorithm that simultaneously optimize both objectives. Simulation results on a set of real-world applications emphasize the robust performance of the proposed approach.

**Keywords**—Green Computing; Distributed Computing; Scheduling; Energy Optimisation;

## I. INTRODUCTION

Power and energy are elemental design considerations across computing solutions, from supercomputers to Cloud Computing (CC). The optimization of the energy consumption in these systems is of great interest due to different problems, such as system performance, operational cost, and environmental issues associated to carbon emissions. In this context, resource management and the development of new scheduling strategies are an active research field.

Efficient and effective task scheduling is fundamental for optimal management of every computing system. Scheduling algorithms commonly used nowadays aim at performance optimization improving the Quality of Service (QoS). However, current trends of rapidly growing energy usage create

a need for a new approach which would take into account also the optimization of the energy consumed by the system.

There are two popular techniques for energy savings which can be directly used at the level of scheduling. The Dynamic Power Management (DPM) consolidates applications on a minimum set of computing resources to maximize the number of resources that can be powered down while maximizing utilization of the used ones [1]. Another widely used technical solution is Dynamic Voltage Scaling (DVS). This technique has been proven to be a promising technique with its demonstrated capability for energy optimization (see [2] and the references given there for more details). DVS enables processors to dynamically changing its working voltage and frequency without stopping or pausing the execution of any instruction. The aim is to reduce power consumption. However, this reduction is achieved at the expense of sacrificing clock frequencies; therefore, longer time will be required to execute a given application.

In the present work, we address the problem of scheduling applications on heterogeneous distributed computing systems with the twin objectives of minimizing simultaneously energy consumption and schedule length. The main contribution is a new multi-objective scheduling algorithm which is based on the multi-start GRASP heuristic framework [3], [4]. The main principle of the new scheduling algorithm is founded on the best-effort idea. That is, schedules with good average performance are generated as an initial step. The schedules are explored in a second phase by a local search that exploits DVS to reduce energy consumption; however, we have designed the local search to improves also the schedule length. The proposed approach is assessed by simulation run on a set of real-world applications. Simulation results showed that this new multi-objective algorithm significantly improves the performance of related approaches.

The organization of the paper is as follow: Section II describes the system, application, energy and scheduling models used in this paper. Section III discusses related work.

Section IV introduces the proposed solution. Experimental results are given in Section V. Section VI concludes the paper.

## II. MODELS

### A. System and Application Models

The underlying execution support considered in this work is a cloud computing system which is a set of resources designed to be allocated ad hoc to run applications. The model we consider is based on the work reported in [5]. In this model, the cloud is assumed to be hosted in a data center which is composed by heterogenous machines. This data center provides a set of services hosted on thousands of high-end computing servers. The need in terms of services or resources (i.e., virtual machines) of an application can be modeled by a task graph. In this graph, an edge between two tasks represents as an inter-service communication.

The cloud computing system consists of a set  $P$  of  $m$  heterogeneous processors. The processors have different processing speed or provide different processing performance in term of MIPS (Million Instruction Per Second). Each processor  $p_j \in P$  is DVS-enabled; that is, it can be operated on a set  $DVS_j$  of voltage and relative speed pairs. Each pair is a supply voltage  $v_k$  and corresponding to it relative speed  $rs_k$ . In this work, we assume that for each processor, a set  $DVS_j$  is random and uniformly distributed among six different sets of DVS (see Table I). We consider that processors consume energy while idling; that is, when a processor is idling it is assumed that the lowest voltage is supplied [6]. Because clock frequency transition overheads take negligible amount of time (e.g. 10  $\mu$ s–150  $\mu$ s), these overheads are not considered in this paper. The inter-processor communications are homogeneous and perform without contention. It is assumed that a message can be transmitted from one processor to another while a task is executed on the recipient processor which is possible in many systems.

We represent the parallel program by a Directed Acyclic Graph (DAG). A DAG,  $G = (T, E)$ , consists of a set  $T$  of  $n$  nodes that correspond to the tasks  $t_i$  of the parallel program and a set  $E$  of  $e$  edges. Each task  $t_i \in T$  has associated basic execution time which is an independent value for each machine. The basic execution time of task  $t_i$  on machine  $p_j$  at maximum speed and voltage (i.e., it corresponds to Level 0 in Table I) is denoted as  $p_{ij}$ . The mean execution time of the task  $t_i$  on processors is  $\bar{p}_i$ . Real execution time  $pr_{ijk}$  of the task  $t_i$  on machine  $p_j$  using relative speed  $rs_k$  is defined as:

$$pr_{ijk} = \frac{p_{ij}}{rs_k}. \quad (1)$$

Each edge  $(t_i, t_j) \in E$ , where  $t_i, t_j \in T$ , represents a precedence constraint as well as inter-task communication. It means that, the output of task  $t_i$  has to be transmitted to task  $t_j$  in order for task  $t_j$  start its execution. The weight on

any edge stands for the communication requirement among the connected tasks. Thus, to every edge  $(t_i, t_j) \in E$  there is an associated value  $c_{ij}$  representing the time needed to transfer data from  $t_i$  to  $t_j$ . However, a communication cost is only required when two tasks are assigned to different processors. That is, the communication cost when tasks are assigned to the same processor can be neglected, i.e., 0. The precedence constraint is however still valid in such case. A task with no predecessors is called an entry task,  $t_{entry}$ , whereas an exit task,  $t_{exit}$ , is one that does not have any successors. Among the predecessors of a task  $t_i$ , the predecessor which completes the communication at the latest time is called the most influential parent (MIP) of the task denoted as  $MIP(t_i)$ . The longest path of a graph is called the critical path (CP). The Earliest Start Time (EST) of and the Earliest Finish Time (EFT) of, a task  $t_i$  on a processor  $p_j$  is defined as:

$$EST(t_i, p_j) = \begin{cases} 0 & \text{if } t_i = t_{entry} \\ EFT(MIP(t_i), p_k) + c_{MIP(t_i), i} & \text{otherwise} \end{cases} \quad (2)$$

$$EFT(t_i, p_j) = EST(t_i, p_j) + p_{ij}, \quad (3)$$

where  $p_k$  is the processor on which  $MIP(t_i)$  is scheduled. Note that the Actual Start Time and Actual Finish Time of a task  $t_i$  on a processor  $p_j$ , denoted as  $AST(t_i, p_j)$  and  $AFT(t_i, p_j)$  can be different from its earliest start and finish times,  $EST(t_i, p_j)$  and  $EFT(t_i, p_j)$ , if the actual finish time of another task scheduled on the same processor is later than  $EST(t_i, p_j)$  [6].

Figure 1 shows a sample task graph and a table with its details. The values presented in the last column of the table are computed based on a frequently used task prioritization method bottom level (*b-level*). Note that, both computation and communication costs are averaged over all nodes and links. The *b-level*( $t_i$ ) is computed recursively by traversing the DAG upward starting from the exit task  $t_{exit}$  as follows (Eq. 4):

$$b\text{-level}(t_i) = \bar{p}_i + \max_{t_j \in succ(t_i)} \{b\text{-level}(t_j) + c_{ij}\}, \quad (4)$$

where  $succ(t_i)$  is the set of immediate successors of  $t_i$  and  $b\text{-level}(t_{exit}) = (\bar{p}_{t_{exit}})$ . The communication to computation ratio (CCR) is a measure that indicates whether a task graph is communication intensive, computation intensive or moderate. For a given task graph, it is computed by the average communication cost divided by the average computation cost on a target system.

### B. Energy Model

We consider the energy model derived from the power consumption model in complementary metal-oxide semiconductor (CMOS) circuits [6]. The power consumption of a CMOS-based microprocessor is defined to be the summation

Table I  
VOLTAGE-RELATIVE SPEED PAIRS [6], [7]

Level	Pair 1		Pair 2		Pair 3		Pair 4		Pair 5		Pair 6	
	Volt. ( $v_k$ )	Rel. Speed (%)	Volt. ( $v_k$ )	Rel. Speed (%)	Volt. ( $v_k$ )	Rel. Speed (%)	Volt. ( $v_k$ )	Rel. Speed (%)	Volt. ( $v_k$ )	Rel. Speed (%)	Volt. ( $v_k$ )	Rel. Speed (%)
0	1.50	100	2.20	100	1.50	100	1.75	100	1.20	100	1.35	100
1	1.20	80	1.90	85	1.40	90	1.40	80	1.15	90	1.25	85.7
2	0.90	50	1.60	65	1.30	80	1.20	60	1.10	80	1.20	71.5
3			1.30	50	1.20	70	0.90	40	1.05	70	1.10	57.1
4			1.00	35	1.10	60			1.00	60	0.9	32.2
5					1.00	50			0.90	50		
6					0.90	40						

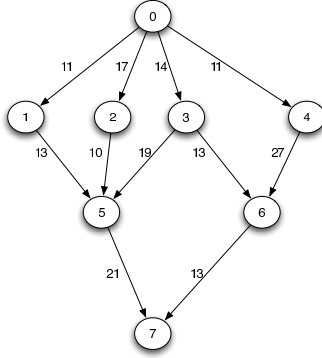


Figure 1. A sample Directed Acyclic Graph with the task numbers  $n_i$  inside nodes and values of  $c_{ij}$  function next to the corresponding edges.

Table II  
A TABLE WITH VALUES OF COMPUTATIONAL COST  $p_{ij}$ , MEAN COMPUTATIONAL COST  $\bar{p}_i$  AND B-LEVEL VALUES FOR EACH TASK.

task	$p_0$	$p_1$	$p_2$	$\bar{p}_i$	b-level
0	11	13	9	11	101.33
1	10	15	11	12	67
2	9	12	14	11.67	63.67
3	12	16	10	12.67	73.67
4	15	11	19	15	79
5	13	9	5	9	42
6	11	15	13	13	37
7	11	15	10	12	12

of capacitive, short-circuit and leakage power (static power dissipation). The most significant factor is the capacitive power, which can be interpreted as the dynamic power consumption [6]. The power consumption  $P_{ij}$  of machine  $m_i$  during the execution of the task  $t_j$  is calculated as follows:

$$P_c = AC_{ef}v^2f, \quad (5)$$

where  $A$  is the number of switches per clock cycle,  $C_{ef}$  is the total capacitance load,  $v$  is the supply voltage, and  $f$  is the machine frequency. Equation 5 indicates the reduction of the supply voltage would be most influential to lower power consumption because it is the dominant factor. The energy consumption of the execution of a precedence-constrained

parallel application used in this work is defined as:

$$E_c = \sum_{i=1}^n AC_{ef}v_i^2f \cdot p_i^* = \sum_{i=1}^n \gamma v_i^2 p_i^*, \quad (6)$$

where  $\gamma = AC_{ef}$  is assumed constant for a given machine;  $v_i$  is the voltage supply value of the processor on which task  $t_i$  is executed, and  $p_i^*$  is the computation cost of task  $t_i$  on the scheduled processor. We also consider energy consumption during idle time. During idle time, processors are automatically scaled to lowest possible voltage level. Energy consumption during idle is defined as [6]:

$$E_i = \sum_{j=1}^m \sum_{idle_{jk} \in IDLE_j} \gamma v_{j,low}^2 I_{jk}, \quad (7)$$

where  $IDLE_j$  is the set of idling slots on machine  $p_j$ ,  $V_{j,low}$  is the lowest supply voltage on  $p_j$ , and  $I_{jk}$  is the amount of idling time for  $idle_{jk}$ . Then the total energy consumption is defined as:

$$E_t = E_c + E_i. \quad (8)$$

### C. The scheduling model

The scheduling problem is the process of allocating a set  $T$  of  $n$  tasks to a set  $P$  of  $m$  processors (without violating precedence constraints) aiming to minimize the schedule length (makespan) with energy consumption as low as possible. The makespan  $C_{max}$  of a DAG represents the time at which the last component of the application finishes execution. It is defined as  $C_{max} = \max AFT(n_{exit})$  after the scheduling of  $n$  tasks in the DAG is completed. Because our scope is on scheduling workflows, we do not consider deadlines as a constraint within our problem definition. However, inclusion of such a constraint will have little or no bearing on our problem formulation.

## III. RELATED WORK

The scheduling problem without energy consideration is NP-hard in its simplest version (without considering communications and homogeneous case). Therefore, many heuristics have been proposed to schedule DAG applications in heterogeneous distributed system environments. A well-known scheduling algorithm in heterogeneous distributed

systems is the Heterogeneous Earliest Finish Time (HEFT) algorithm [8]. HEFT is founded on the list scheduling principle. It maintains a list of all tasks of a given graph according to their priorities. It consists in two phases. In the first phase of the algorithm a ready task is selected from the list based on its priority. The task with highest priority is selected. This process corresponds to the *task prioritizing* or *task selection* phase. Then, a suitable processor that minimizes a predefined cost function is selected (i.e., the *processor selection* phase). HEFT is highly competitive in that it generates a schedule length comparable to other scheduling algorithms, with a low time complexity ( $O(n \log n + (e + n)m)$ ).

The most common approach that energy-aware scheduling algorithms exploit is the DVS technique [6], [7], [9], [10]. In [12], different scheduling algorithms using the concept of slack sharing among DVFS-enable processors were proposed. The rationale behind the algorithms is to utilize idle (slack) time slots of processors lowering supply voltage (frequency/speed). This technique is known as slack reclamation. These slack time slots occur, due to earlier completion (than the worst-case execution time) and/or dependencies of tasks. [11] gives a formulation of energy aware scheduling algorithm and a detailed discussion of slack time computation. This scheduling algorithm also concerns reducing voltages during the communication phases between parallel jobs on homogeneous processors. Ref. [6] proposes heuristics that are devised with relative superiority as a novel objective function, which takes into account energy and performance. The algorithm consists in two phases. In the first phase a linear function of both objectives is proposed and a schedule is computed based on this function. Since each scheduling decision that makes tends to be confined to a local optimum, another energy reduction technique (the second phase) is incorporated into the energy reduction phase.

Concerning multi-objective approaches, the authors in [10] proposed a bi-objective genetic algorithm which is improved with the first phase of the algorithm developed in [6]. Precedence-constrained task graphs were considered. DVFS was used to optimize energy. The proposed approach computes a set of solutions instead of only one. The work reported in [10] has been extended in [5] to a parallel model of their approach. The parallelization is based on the island parallel model and multi-start parallel model.

#### IV. PROPOSED MULTI-OBJECTIVE GRASP

##### A. Multi-Objective Optimization and GRASP

Given a set of  $M$  objectives  $f_1, f_2, \dots, f_M$  to be minimized, solution  $s_1$  weakly dominates solution  $s_2$ , denoted  $s_1 \preceq s_2$ , whenever :

- The solution  $s_1$  is no worse than  $s_2$  in all objectives or  $f_i(s_1) \leq f_i(s_2) \forall i \in 1, 2, \dots, M$ .

- If, in addition, there exists a  $j \in 1, 2, \dots, M$  such that  $f_j(s_1) < f_j(s_2)$ , then  $s_1$  strictly dominates  $s_2$ , denoted  $s_1 \prec s_2$ .

Given a set of solutions,  $S$ , a solution  $a \in S$  is *non-dominated* if there are no solutions  $s \in S$  that strictly dominate  $a$ . If  $S$  is a set of all feasible solutions, a non-dominated solution is *Pareto-optimal* [13]. The set of solutions which could satisfy the needs of a decision maker is the set of all Pareto-optimal solutions. These solutions represents the possible trade-offs, as each solution in the Pareto set, is non-dominated in Pareto sense. Plotting the positions of the Pareto-optimal solutions in the objective space results in the *Pareto-front*. To facilitate the decisions of the decision maker, a multi-objective algorithm should provide a set of solutions that closely approximates the Pareto-front.

Greedy Randomized Adaptive Search Procedure (GRASP) is an iterative or multi-start randomized search technique for producing good-quality solutions for combinatorial optimization problems [3], [4]. The iterative GRASP framework consists of two phases: greedy construction and local search or post-processing. The construction phase build a feasible solution which is improved in the post-processing phase by investigating its neighborhood with a local search procedure until a local minimum is found. This process is repeated a number of iterations to search possible optimal solutions. The best overall solution is kept as the result. A GRASP is terminated when the specified criterion is satisfied, for example, after completing a certain number of iterations. In the first phase, GRASP constructs a solution one element at a time. Candidate elements are assessed using a greedy evaluation function and the best elements form an adaptive *restricted candidate list* (RCL). Two major mechanism can be used to generate the RCL list, *value-based* and *cardinality-based* mechanism. In the value-based case, the RCL is associated with a parameter  $\alpha \in [0, 1]$  and a given threshold. All the candidates whose incremental cost is no greater than the threshold value are recorded into the RCL list. The cardinality-based mechanism records the  $k$  best rated solution elements, where  $k$  is a given parameter.

##### B. Energy-aware scheduling GRASP

We propose a new multi-objective GRASP solution based on the best-effort idea and following the principle of list scheduling heuristics. That is, in the construction phase the algorithm search for good schedules (i.e., with minimum makespan) using maximum voltage. The aim of this phase is to exploit the heterogeneity of resources to optimize performance (best-effort idea). Then, a voltage scaling algorithm is used to improve the energy consumption without increasing the schedule length of the constructed solution. In the local search phase the neighborhood of the constructed solution is visited to construct the set of non-dominated solutions. **Algorithm 1** presents the pseudo-code for the multi-objective

energy-aware scheduling GRASP. The variable  $N$  represents the number of iterations of GRASP, the variable  $I$  represents the number of times the local search is executed per iteration. The algorithm starts by initializing (line 2) the *bestfront* which represents the Pareto-front. A list  $L$  of tasks priorities is constructed (line 2). This list is used in the construction phase to determine the order in which tasks will be evaluated to help in the construction of the solution. The aim is to reduce time complexity in the construction phase. This list is computed only one and is not modified through the optimization process. The **OneGeneration**( $G = (T, E), L, I$ ) function (line 5) constructs a new front at each iteration of Algorithm 1. The function calls the construction phase, the slack reclamation algorithm and local search phase. After that, the set of non-dominated solutions is modified by adding any new non-dominated solutions (line 6), hence any solutions that become dominated are removed from the non-dominated set (line 7).

**Algorithm 1.** Multi-objective GRASP.

```

1: function MOGRASP( $G = (T, E), N, I$ )
2:    $bestfront := \emptyset$ 
3:    $L := \mathbf{ComputeListPriority}(G = (T, E))$ 
4:   for  $x:=1$  to  $N$  do
5:      $newFront := \mathbf{OneGeneration}(G = (T, E), L, I)$ 
6:      $bestFront := bestFront \cup newFront$ 
7:     Remove dominated solutions from  $bestFront$ 
8:   end for
9: return  $bestFront$ 
10: end function
11: function OneGeneration( $G = (T, E), L, I$ )
12:    $solution := \mathbf{ConstructSolution}(G = (T, E), L)$ 
13:    $solution' := \mathbf{VoltageScaling}(solution)$ 
14:    $front := \mathbf{MOLocalSearch}(solution', I)$ 
15: return  $front$ 
16: end function

```

**Algorithm 2** constructs the list of priorities based on the  $b$ -level metric (line 2) and the List is sorted in decreasing order of priority (line 3).

**Algorithm 2.** Construct List of Priorities.

```

1: function ComputeListPriority( $G$ )
2:   Calculate the priority of each task according to the
    $b$ -level value using Eq. 4
3:   Sort the tasks in a priority list  $L$  by decreasing order
   of  $b$ -level
4: return  $L$ 
5: end function

```

**Algorithm 3** computes a feasible solution based on the List scheduling principle. A feasible solution for the investigated problem is required to respect precedence constraints and every task is scheduled once and only once on the pro-

cessors. The algorithm uses a value based RCL mechanism. It starts by selecting a ready task (i.e. the task at the top of the list) from the list  $L$  (line 4). The scheduler estimates the makespan increase for each ready task, that is, it computes the EFT value on each processor (line 7). The makespan increase of a task  $t_i$  on a processor  $p_j$  is the increase of the execution length to the current completion time if task  $t_i$  is allocated on  $p_j$ . The greedy algorithm selects a task assignment randomly from the task and resource pair whose makespan increase is less than the threshold  $minI + \alpha(maxI - minI)$  (line 10 to 15), where  $minI$  and  $maxI$  are the lowest and highest makespan increase found respectively. The variable  $\alpha$  is a parameter to determine how much variation is allowed for creating RCL for each task and  $\alpha \in [0, 1]$ . Once a feasible solution is constructed, we apply a voltage scaling algorithm to reduce the energy of the current schedule without allowing the degradation of the makespan.

The local search is applied into the neighborhood of the solution after the construction phase and voltage scaling. We propose a local search (**Algorithm 4**) that adopts DVS technique to reduce energy consumption, however makespan is improved by allocating tasks to different current allocation (i.e., processor). This local search is an iterative search process in which the local search direction is randomly selected. However, we have designed the local search bearing in mind two basic considerations of local search *diversification vs intensification*. In the first step of the local search procedure, the front is initialized with the feasible solution computed in the construction phase (line 2). The algorithm retains any new non-dominated solutions and eliminates those that are dominated. A task is selected from the current initial solution at random (line 4). A simple neighborhood point of a schedule in the solution space is another schedule that is obtained by transferring a task from a processor to another processor (line 7) and changing a voltage and speed level to another operating point (i.e. level) of the selected processor (line 8). Changing the current allocation of tasks explores possible makespan improvement while changing a voltage and speed level explores energy gain (line 9). The new solution is evaluated (line 10), if the neighbor of the initial solution is non-dominated, the local search moves to the refined solution point (line 12 and 13). Otherwise, the movements are not allowed and task  $t_i$  and voltage  $v_k$  are moved back to their original processor and operating point, and another search direction is randomly selected. The constant *MAXSTEPS* in Algorithm 4 is defined to limit the number of steps so that only *MAXSTEPS* different current allocation and voltages levels are explored (intensification). This process is iteratively repeated  $I$  times, so that  $I$  tasks are inspected at each call of the local search procedure (diversification).

**Algorithm 3.** Construction phase function.

```

1: function ConstructSolution( $G = (T, E), L'$ )
2:   solution :=  $\emptyset$ 
3:   while  $L' \neq \emptyset$  do
4:     Select ready task,  $t_i$ , from the top of  $L'$ 
5:     pairs :=  $\emptyset$ 
6:     for all  $p_j \in P$  do
7:       Compute EFT( $t_i, p_j$ )
8:       pairs := pairs  $\cup (t_i, p_j)$ 
9:     end for
10:    minI := minimum EFT over pairs
11:    maxI := maximum EFT over pairs
12:    RCL := pairs whose EFT < minI +  $\alpha(maxI - minI)$ 
13:    ( $t'_i, p'_j$ ) := select a pair ( $t_i, p_j$ ) at random from RCL
14:    Remove ready task,  $t_i$ , from  $L'$ 
15:    solution := solution  $\cup (t'_i, p'_j)$ 
16:  end while
17: return solution
18: end function

```

**Algorithm 4.** Multi-objective Local Search Function.

```

1: function MOLocalSearch(solution, I)
2:   front := solution
3:   for iter := 1 to I do
4:     Select a task  $t_i$  at random from solution
5:     for searchstep := 1 to MAXSTEPS do
6:        $\triangleright$  Next step explores possible makespan improvement
7:       Select a processor  $p_k \neq p_j$  at random  $\triangleright p_j$ 
         is the current location of task  $t_i$ 
8:       Select ( $v_k, rs_k$ ) from the corresponding set
         of voltage and relative speed of  $p_k$  randomly  $\triangleright$  DVFS
         technique to explore energy improvement
9:       Allocate  $t_i$  on  $p_k$  with voltage  $v_k$  and relative
         speed  $rs_k$ 
10:      solution' := Compute current EFT and En-
         ergy
11:      if solution' is not dominated by any member
         of front then
12:        front := front  $\cup$  solution'
13:        solution := solution'
14:      end if
15:    end for
16:  end for
17:  return front
18: end function

```

### C. Complexity Analysis

The upper bound complexity of the developed algorithm (**Algorithm 1**) is as follows. **Algorithm 2** starts by computing the b-level priority for each task. The upper bound of this algorithm is  $O(e + n)$ . The sorting of line 3 can be

done in  $O(nlgn)$ . Therefore, the complexity of Algorithm 2 is  $O(e + n + nlgn)$ . Algorithm 1 calls OneGeneration function. This function calls Algorithm 3, a voltage scaling algorithm and Algorithm 4. The complexity of **Algorithm 3** is  $O(e + n)m$ . It is based on the list scheduling principle. The voltage scaling algorithm scales the supply voltage of a processor. Assuming we have the  $m$  processors with  $TS$  time slots, thus the upper bound of this algorithm is  $O(mTS)$ . **Algorithm 4** explores  $I$  tasks and selects  $MAXSTEPS$  processor and voltage pairs. At each iteration of the random local search we compute a new schedule (line 10). Line 9 moves a task  $t_i$  to a processor  $p_k$  and voltage  $v_k$ . Each move takes  $O(e)$  time to compute the schedule length. Line 11 compares the new solution with the current front. This operation takes  $K$  steps, where  $K$  is the number of non-dominated solutions in the current front. Hence, the upper bound complexity of Algorithm 4 is  $O(IMAXSTEPS(e + N))$ . To identify the non-dominated set of solutions (line 6) in Algorithm 1,  $K$  non-dominated solutions in the current front are compared  $ND$  times with the solutions in the Pareto front, where  $ND$  is the maximum number of non-dominated solutions we allow in the Pareto front. Thus the upper complexity to construct the Pareto front is  $O(KND)$ .

The loose upper bound of the new developed algorithm is :

$$O(|e| + |n| + |n|lg|n| + N(|m|(e + n) + |m||TS| + |I||MAXSTEPS|(e + n) + |K||ND|)).$$

## V. EXPERIMENTS

We report experimental results to validate the new proposed approach. We compare the multi-objective GRASP against HEFT. We use HEFT because it is one of the most practical and applicable heuristics for DAGs in distributed heterogeneous computing systems. We have also implemented HEFT with Slack Reclamation. It is an extension of the work proposed in [11]. In [11] proposed a list scheduling algorithm with DVFS to optimize energy without performance degradation. The idea is to use idle time mainly due to precedence constraint and communication delays. DVFS is applied to no critical task. A task is critical if it is in the critical path of the schedule. However, the authors considered a homogeneous set of processors. We extend to heterogeneous case considering HEFT as the list scheduling algorithm. As far as we are concerned no HEFT + DVFS approach has been developed, despite its simplicity and effectiveness.

We show in Figure 2 a sample Pareto front computed by the multi-objective GRASP on the instance provided as example in Figure 1. The number of GRASP iterations for this small example was fixed to 50. We have used  $\alpha \in [0, 0.2]$ . We show this example to illustrate the basic multi-objective definitions provided in Section IV. We can observe that HEFT+DVFS weakly dominates the HEFT

solution. The curve represents the Pareto front and crossed points represent the set of Pareto-optimal solutions computed by our algorithm.

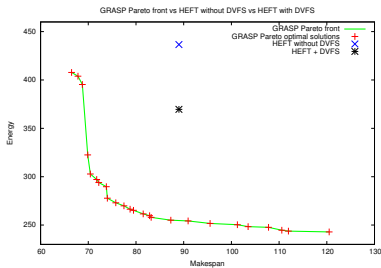


Figure 2. A sample Pareto front computed by the new proposed solution.

### A. Experimental Settings

The performance of the multi-objective GRASP was evaluated on a set of structured real-world applications. The four real-world parallel applications used for our experiments were the robot and the sparse matrix solver from the *Standard Task Graph Set* (STG) [14], the Laser Interferometer Gravitational Wave Observatory (LIGO) application and Gaussian Elimination (GE) that has been generated synthetically. The GE application represents a graph for solving standard matrix multiplication for which the DAG structure is known and dependent on the input size. Table III describes the main characteristics for these applications: instances size, edges amount and the ratio between tasks and edges (ETR). ETR gives information on the degree of parallelism. For GE elimination we have varied the size of the instance: 42, 52, 63, 75 and 88 tasks. The computational costs of the tasks in each application were generated as described in [8]. We fixed the parameter  $\beta$  to 1. Parameter  $\beta$  is basically the heterogeneity factor for processor speeds. A high percentage value (i.e., a percentage of 1) causes a significant difference in a task’s computation cost among the processors. Additionally, for each graph we have varied the CCR ratio. We have generated five CCRs (0.1, 0.5, 1, 5, 10) for each graph and instances size. The execution of the applications is performed on 8, 16 and 32 processors.

The total number of multi-objective GRASP iterations was  $N = 200$ , the maximum number of movements during the local search was fixed to  $I = 60$ ,  $\alpha \in [0, 0.2]$ . These parameters were set by experiments. We have decided to use an interval for  $\alpha$  close to 0 based on the best-effort idea. That is, to perform the construction phase like a list scheduling algorithm with some degree of randomness.

### B. Experimental Results

Let us remark that the new proposed approach computes a set of non-dominated solutions and the two reference approaches provides only one solution. To compare the new multi-objective GRASP with HEFT and HEFT+DVFS, we have considered the methodology proposed in [10]. To

Table III  
EMPLOYED INSTANCES AND THEIR MAIN CHARACTERISTICS

Application	# of Tasks	# of Edges	ETR
LIGO	76	132	1.73
Robot control	88	131	1.48
Sparse matrix	96	128	0.69
GE	42	68	1.61
	52	86	1.65
	63	106	1.68
	75	128	1.70
	88	152	1.72

determine the contribution of the new approach, we compare the solution provided by HEFT and HEFT+DVFS to only one solution of the Pareto set provided by the new approach. However, the proposed algorithm provides a set of Pareto solutions to the decision maker instead of one solution.

The methodology used to validate the approach follows the next steps. For each instance, a first resolution is computed with HEFT, and HEFT+DVFS to obtain solution  $s_{HEFT}$  and  $s_{HEFT+DVFS}$ . A second resolution is done with the new proposed approach to generate a set of Pareto solutions. Only one Pareto solution  $s_{GRASP}$  is selected from the set of Pareto solutions. This solution is the closest to  $s_{HEFT}$  and  $s_{HEFT+DVFS}$  in the sense of Euclidean distance. Finally, a comparison is done between  $s_{HEFT}$  and  $s_{GRASP}$ , and  $s_{HEFT+DVFS}$  and  $s_{GRASP}$ . The  $s_{GRASP}$  closest solution to  $s_{HEFT}$  can be different to the  $s_{GRASP}$  closest solution to  $s_{HEFT+DVFS}$ .

The results for the simulations are displayed from Table IV, Table V, Table VI, Table VII. These results show that the new proposed approach improves on average the results obtained by HEFT and HEFT+DVFS. The gain in the makespan is 7.77% and in the energy is 15.51% regarding HEFT. The improvement in the makespan is 7.78% and in the energy the gain is 10.47% against HEFT+DVFS. Tables IV, V, VI, and VII provide detailed results. These tables respectively show the gain computed by the proposed solution according to the number of processors, CCRs, kind of application and according to the size of the instance for the GE application. From these tables we can observe that the factor that impact the most in the behavior of the new approach is CCR. We must note that a low CCR means that the application is considered computation-intensive. On the contrary, for a high value of CCR the application is communication intensive. In the former, is harder to exploit the DVFS technique than for communication-intensive applications. However, in both cases the new algorithm is able on average to provide better schedules than the reference approach which is interesting given the good performance behavior of HEFT.

## VI. CONCLUSION

In this paper, we have designed a new multi-objective scheduling algorithm for performance and energy optimiza-

Table IV  
GAIN ACCORDING TO THE NUMBER OF PROCESSORS

Num. of Procs	Gain over HEFT (%)		Gain over HEFT+DVFS (%)	
	Makespan	Energy	Makespan	Energy
8	7.41	15.0	7.28	9.37
16	7.56	15.63	7.51	9.73
32	8.65	15.32	8.64	11.52

Table V  
GAIN ACCORDING TO CCR

CCR	Gain over HEFT (%)		Gain over HEFT+DVFS (%)	
	Makespan	Energy	Makespan	Energy
0.1	1.61	10.22	1.57	4.19
0.5	5.27	12.84	5.18	7.12
1	6.64	13.97	6.54	8.76
5	11.62	19.27	11.53	14.50
10	14.58	19.83	14.52	16.41

Table VI  
GAIN ACCORDING TO REAL WORLD APPLICATIONS

Appl.	Gain over HEFT (%)		Gain over HEFT+DVFS (%)	
	Makespan	Energy	Makespan	Energy
LIGO	8.13	15.56	8.13	13.03
ROBOT	8.12	17.78	8.07	9.94
SPARSE	5.24	16.35	5.24	15.20

Table VII  
GAIN ACCORDING TO THE DAG SIZE FOR GE APPLICATION

No. of Tasks	Gain over HEFT (%)		Gain over HEFT+DVFS (%)	
	Makespan	Energy	Makespan	Energy
42	9.05	15.55	9.0	10.28
52	8.97	15.6	8.95	9.56
63	8.24	15.06	8.05	8.9
75	7.59	14.69	7.36	8.09
88	8.33	11.25	8.19	6.56

tion on cloud computing systems. The new algorithm is based on a GRASP framework. The algorithm simultaneously optimizes both conflicting objectives and provides a set of non-dominated solutions or Pareto-optimal solutions. The proposed algorithm is based on the best-effort idea in the construction phase. The aim is to exploit the heterogeneity of resources to provide good schedules with minimum makespan. The second phase of the algorithm exploits DVFS technique to reduce energy, however we have designed the local search to improve also the makespan. The new proposed approach has been evaluated on a set of real-world applications and synthetic graphs showing its potential to solve the problem.

We plan to extend the proposed work. First, for the construction phase we plan to construct a front of solutions instead of construct only one solution at each iteration of the algorithm. We also consider to investigate more local search algorithms. Concerning the model we plan to consider virtual machines.

#### ACKNOWLEDGMENT

The authors would like to acknowledge the funding from Luxembourg FNR in the framework of GreenIT project

(C09/IS/05).

#### REFERENCES

- [1] Benini, L., Bogliolo, A., De Micheli, G.; A survey of design techniques for system-level dynamic power management, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 3, pp. 299–316, June 2000.
- [2] Valentini, G., Lassonde, W., Khan, S., Min-Allah, N., Madani, S., Li, J., Zhang, L., Wang, L., Ghani, N., Kolodziej, J., Li, H., Zomaya, A.Y., Xu, C.-Z., Balaji, P., Vishnu, A., Pinel, F., Pecero, J., Kliazovich, D., and Bouvry, P.; An overview of energy efficiency techniques in cluster computing systems, *Cluster Computing*, pp. 1–13, issn 1386–7857, doi: 10.1007/s10586-011-0171-x.
- [3] Feo T.A., Resende, M.G.C.; Greedy randomized adaptive search procedures, *J Glob Optim*, vol. 6, pp.109–133, 1995.
- [4] Festa, P. and Resende, M. G.C.; GRASP: basic components and enhancements, *Telecommunication Systems*, vol. 44(3), pp. 253–271, 2010.
- [5] Mez maz, M., Melab, N., Kessaci, Y., Lee, Y.C., Talbi, E.-G., Zomaya, A.Y., Tuyttens, D.; A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems, *J of Parallel and Dist Computing*, In Press.
- [6] Lee, Y. C. and Zomaya, A. Y.; Energy Conscious Scheduling for Distributed Computing Systems under Different Operating Conditions, *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1374–1381, Aug. 2011
- [7] Rizvandi, N. B., Taheri, J., Zomaya, A. Y. and Lee, Y. C.; *Linear Combinations of DVFS-Enabled Processor Frequencies to Modify the Energy-Aware Scheduling Algorithms*, IEEE/ACM CCGRID’10, pp. 388–397, Australia, 2010
- [8] Topcuoglu, H., Hariri, S. and Wu, M. Y.; Performance-effective and Low Complexity Task Scheduling for Heterogeneous Computing, *IEEE Trans. Parallel Distrib. Syst.*, Vol. 13, Issue 3, Pages 260-274, 2002.
- [9] Khan, S.U., and Ahmad, I.; A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids, *IEEE Trans. Parallel Distrib. Syst.*, 20(3): 346–360, 2009.
- [10] Mez maz, M., Lee, Y.C., Melab, N., Talbi, El-G. and Zomaya, A.Y.; *A bi-objective hybrid genetic algorithm to minimize energy consumption and makespan for precedence-constrained applications using dynamic voltage scaling*, IEEE CEC, pp. 1–8, Spain, 2010.
- [11] Wang, L., von Laszewski, G., and Dayal, J.; *Towards Energy Aware Scheduling for Precedence Constrained Parallel Tasks in a Cluster with DVFS*, IEEE/ACM CCGRID’10, pp. 368–377, Australia, 2010.
- [12] Zhu, D., Mosse, D., Melhem, R., Power-aware scheduling for AND/OR graphs in real-time systems, *IEEE Trans. Parallel Distrib. Syst.*, Vol. 15, No. 9, pp. 849–864, 2004. 69–75. IEEE Computer Society, Arica (2004).
- [13] Reynolds, A. and de la Iglesia, B.; A multi-objective GRASP for partial classification. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 13(3), pp. 227–243, Springer, 2008.
- [14] Tobita, T., Kasahara, H.; A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, *J Scheduling*, Vol. 5, Issue 5, pp. 379–394, 2002.