Energy efficient scheduling in heterogeneous systems with a parallel multiobjective local search

Santiago Iturriaga, Sergio Nesmachnow, Dorronsoro Bernabe, Pascal Bouvry

Computing and Informatics, Vol. 32, 2013, 273–294

ENERGY EFFICIENT SCHEDULING IN HETEROGENEOUS SYSTEMS WITH A PARALLEL MULTIOBJECTIVE LOCAL SEARCH

Santiago Iturriaga, Sergio Nesmachnow

Universidad de la República, Montevideo, Uruguay
e-mail: {siturria, sergion}@fing.edu.uy

Bernabé Dorronsoro

LIFL, University of Lille 1, France
e-mail: bernabe.dorronsoro@lifl.fr

Pascal Bouvry

University of Luxembourg, Luxembourg
e-mail: pascal.bouvry@uni.lu

Abstract. This article introduces ME-MLS, an efficient multithreading local search algorithm for solving the multiobjective scheduling problem in heterogeneous computing systems. We consider the minimization of both the makespan and energy consumption objectives. The proposed method follows a fully multiobjective approach, applying a Pareto-based dominance search that is executed in parallel by using several threads. The experimental analysis demonstrates that the new multithreading algorithm outperforms a set of fast and accurate two-phases deterministic heuristics based on the traditional MinMin. The new ME-MLS method is able to achieve significant improvements in both makespan and energy consumption objectives in reduced execution times for a large set of testbed instances, while exhibiting a near linear speedup behavior when using up to 24 threads.

Keywords: Scheduling, local search, multithreading, heterogeneous computing

1 INTRODUCTION

Distributed computing platforms have grown at a fast pace. Nowadays, they gather many heterogeneous computing (HC) resources located worldwide. In the last decade, the novel *grid computing* systems emerged as useful providers for the computing power needed to tackle complex problems in many application domains [7].

In current distributed HC/grid systems, efficiently allocating tasks to be executed on available resources (*scheduling*) is a key problem to take full advantage of the computing power. The goal of the scheduling problem is to assign tasks to the computing resources by satisfying specific efficiency criteria, usually related to the total execution time of a bunch of tasks (*makespan*), but frequently also considering other relevant objectives such as resource utilization, quality of service, etc.

Traditional scheduling problems are NP-hard [8], thus classic exact methods are not useful in practice to solve large instances of the problem. Many deterministic scheduling heuristics have been proposed, but in general they do not scale appropriately when solving large dimension instances, also demanding long execution times. Thus, non-deterministic heuristics and metaheuristics have been used to tackle scheduling problems, and they are shown to compute efficient schedules in reasonable timespans for both small and large dimension instances [25, 26, 30, 32].

Recently, energy consumption has become a major concern in distributed/high performance computing. Processors are the main consumers of energy in such systems, and frequently they also offer the most flexible options for energy management, by applying dynamic voltage scaling (DVS), dynamic power management, slack sharing and reclamation, etc. [11, 13, 33]. Reducing processors consumption is a great challenge, and researchers currently focus on the development of energy-aware scheduling algorithms for HC/grid systems [17].

In this article, we propose, implement, and evaluate ME-MLS, a highly efficient multiobjective local search metaheuristic to find accurate tradeoff solutions to the multiobjective scheduling problem of minimizing the makespan and the energy consumption in HC/grid systems. ME-MLS is a population-based method that follows a fully multiobjective approach; it does not optimize an aggregated function of the problem objectives, but uses a Pareto-based dominance analysis instead. The ME-MLS population is size bounded, thus, two different replacement policies are evaluated:

1. a fast greedy ad-hoc replacement algorithm, and
2. the Adaptive Grid Archiving (AGA) algorithm proposed by Knowles et al [14].

Both ME-MLS variants are compared versus well-known deterministic heuristics over a large set of instances. The experimental results show that ME-MLS is able to compute accurate tradeoff schedules in short execution times.

The manuscript is organized as follows. Section 2 introduces the multiobjective energy-aware scheduling problem in HC systems and a review of related work. The details about the design and implementation of the new multiobjective local search

method is presented in Section 3. Experimental evaluation of the proposed method over a large set of problem instances is described in Section 4. Finally, Section 5 presents the conclusions and formulates the main lines for future work.

2 ENERGY-AWARE SCHEDULING IN HC SYSTEMS

This section introduces the makespan-energy heterogenous computing scheduling problem (ME-HCSP) and a review of related work.

2.1 Problem Formulation

An HC system is composed by a set of heterogeneous computers (usually called *machines*), having different computing power and energy consumption demand, depending on the hardware features.

In this work, we address the problem of scheduling a set of non-dependent tasks on an HC system. In this model, a task is considered as atomic workload unit, so it cannot be divided, nor interrupted after it is assigned to a machine (the scheduling problem follows a *non-preemptive* model). The execution times of any task and the energy consumption to perform it vary from one machine to another. Thus, those machines that are able to execute tasks quickly and with low energy consumption will have a strong demand in the system.

Scheduling problems usually target the minimization of the *makespan*, to reduce the time spent to execute a bunch of tasks. The makespan is defined as the time spent since the first task begins execution to the moment when the last task is completed [19]. The ME-HCSP aims at minimizing the makespan together with the energy consumption in HC systems.

The mathematical model of ME-HCSP considers the following elements:

- An HC system composed of a set of heterogeneous machines $P = \{m_1, \ldots, m_M\}$; each machine performing at a certain processing speed and energy consumption.
- A collection of tasks $T = \{t_1, \ldots, t_N\}$ to be executed on the system.
- An *execution time function* $ET : T \times P \to \mathbf{R}^+$, where $ET(t_i, m_j)$ is the time required to execute task t_i on machine m_j.
- An *energy consumption function* $EC : T \times P \to \mathbf{R}^+$, where $EC(t_i, m_j)$ is the energy required to execute task t_i on machine m_j, and $EC_{IDLE} : P \to \mathbf{R}^+$, being $EC_{IDLE}(m_j)$ the energy that machine m_j consumes in idle state.

The ME-HCSP proposes to find a schedule f that simultaneously minimizes the *makespan* (defined in Equation (1)), and the *energy consumption* (defined in Equation (2)), which accounts for both the energy required for tasks execution and the energy that each machine spends in idle state until all tasks are finished; ($CP(M_j)$ is the completion time of machine j, i.e., the time when it finishes all its assigned tasks). The energy for executing a given task depends on its execution time

$ET(t_i, m_j)$, but the two objectives are in conflict in HC systems, since fast machines usually consume more energy than the slower ones.

$$\max_{m_j \in P} \sum_{\substack{t_i \in T: \\ (t_i, m_j) \in f}} ET(t_i, m_j) \tag{1}$$

$$\sum_{\substack{t_i \in T: \\ (t_i, m_j) \in f}} EC(t_i, m_j) + \sum_{m_j \in P} (makespan - CP(M_j)) EC_{IDLE}(m_j) \tag{2}$$

In the previous ME-HCSP formulation, all tasks can be performed disregarding the execution order. This kind of programs are frequent in e-Science applications over HC/grid computing, such as Single-Program Multiple-Data applications used for multimedia processing, data mining, parallel domain decomposition of numerical models for physical phenomena, etc.

The independent tasks model also arises when different users submit their (independent) tasks to execute in grid and volunteer-based computing services, such as WLCG, Teragrid, Berkeley's BOINC, Xgrid [3], where non-dependent domain decomposition applications are often submitted for execution. Thus, the ME-HCSP faced in this work is relevant in realistic distributed HC/grid environments.

2.2 Related Work

The main trend in energy-aware schedulers proposed in the related literature is to apply energy management methods within the computing elements. The most used strategies for energy management are dynamic voltage scaling (DVS), dynamic power management, and slack sharing/reclamation methods [11, 13].

Kim et al. [12] introduced several online/batch dynamic heuristics for scheduling tasks with priorities and deadlines in an ad-hoc grid with limited battery capacity, using DVS for power management. The batch dynamic heuristics based on the classic MinMin scheduler [22] performed best, but they required significantly longer time. Li et al. [20] also proposed an energy-aware scheduler based on the MinMin heuristic to reduce energy consumption using dynamic power management.

In 2009, Kahn and Ahmad [11] developed an energy-aware grid scheduler based on the concept of Nash Bargaining Solution from cooperative game theory. The scheduler was applied for planning in infrastructures with DVS-enabled machines.

Another cooperative approach to solve the energy-aware scheduling problem was introduced by Mezmaz et al. [24], which applied a cooperative parallel bi-objective hybrid genetic algorithm (GA), improved with the energy-aware heuristics previously proposed by Lee and Zomaya [18].

The work by Pecero et al. [29] applied a bi-objective Greedy Randomized Adaptive Search Procedure (GRASP) scheduler. The proposed method builds a feasible solution using a greedy evaluation function, and uses a post-processing bi-objective local search method applying DVS to improve the solution quality and to generate a set of Pareto solutions.

A two-phase energy-aware heuristic for grid scheduling was proposed by Pinel et al. [31], by first applying the MinMin heuristic to find schedules with good makespan, and a local search in the second phase. The method allowed to find schedules with similar quality to those computed by a cellular evolutionary algorithm (EA), while reducing the execution time, and significantly improving the MinMin schedules.

Kessaci et al. [10] proposed two multiobjective parallel evolutionary algorithms enhanced with energy-aware scheduling heuristics for tasks with dependencies, combining makespan and energy consumption in a unique function and using explicit DVS for energy management.

Kolodziej et al. [16] developed an EA framework for designing energy-aware schedulers, using DVS for energy reduction. The problem model in that work considered bag-of-tasks applications and a bi-objective scheduling problem minimizing makespan and energy consumption. The article by Lindberg et al. [21] introduced a set of eight heuristics, including two EAs for the energy-aware grid scheduling problem subject to deadline and memory constraints.

The approach we describe in this article does not explicitly apply DVS or other energy management methods. Instead, we propose a scheduler that computes the energy consumption by considering the maximum and minimum energy consumption provided by the processor technology, which accounts for hardware-embedded energy saving features. The efficiency of these technologies is model dependent, since hardware-embedded energy saving features are proprietary technologies (e.g. AMD provides the Optimized Power Management technology in the Opteron processor family, Intel provides the SpeedStep technology in Xeon/i7 processor family, etc.). In our approach, a machine with assigned workload is assumed to operate at its peak performance, consuming the maximum specified energy. Conversely, a machine with no workload is assumed to be in idle state with the embedded energy saving technology keeping the energy consumption at its minimum specified value.

3 A MULTITHREADING ENERGY-AWARE HC SCHEDULER

This section describes the newly proposed multithreading local search algorithm to solve the energy-aware scheduling problem in HC systems.

3.1 ME-MLS Design

ME-MLS is a population-based local search algorithm, which maintains a population of non-dominated (*elite*) solutions in order to avoid biasing the search toward any of the two objectives of the energy-aware scheduling problem. A shared-memory parallel implementation is proposed for ME-MLS, by using a pool of threads. Each thread is a peer that performs a search, and no thread acts in a master role.

Algorithm 1 presents the logic of each ME-MLS thread. Schedules are initialized using a randomized version of the Minimum Completion Time (MCT) scheduling

Algorithm 1 Peer threads in the ME-MLS algorithm

```
1: initialize_solution(population, thread_index)
2: initialization_barrier()
3: while not stop_criteria do
4:    lock_population()
5:    s ← select_random_schedule(population)
6:    s' ← clone(s)
7:    unlock_population()
8:    search_strategy ← select_strategy(makespan, energy, or random)
9:    max_iterations ← random(1, THREAD_IT)
10:   while not end_search do
11:      for i = 0 → max_iterations do
12:         local_search(search_strategy, s')
13:      end for
14:      trylock_population()
15:      if locked then
16:         check_Pareto_dominance(population, s')
17:         end_search ← true
18:         unlock_population()
19:      else
20:         max_iterations ← random(1,THREAD_IT/REWORK_FACTOR)
21:         end_search ← false
22:      end if
23:   end while
24: end while
```

heuristic [22] (line 1). Each thread iteratively applies a local search (lines 11 to 13) on a clone s' of a randomly selected schedule s (line 6). Newly found non-dominated schedules are inserted into the population (line 16). Since each thread has to exclusively lock the population to insert a new schedule (lines 14 to 18), they apply an additional number of operations (bounded by the `REWORK_FACTOR` parameter – computed in line 20) waiting for the population lock to be released. After the new schedule is safely inserted into the population (line 16), the thread repeats the whole process (lines 3 to 23) to keep improving non-dominated schedules from the population. Figure 1 shows the steps performed by ME-MLS algorithm.

3.2 Replacement Algorithm

The elite population in ME-MLS is limited in size, so a replacement technique is applied when the maximum size is reached. We evaluated two replacement methods: a fast greedy ad hoc technique, and the Adaptive Grid Archiving technique.

Fast greedy ad hoc replacement. This technique makes use of heuristic knowledge regarding the makespan-energy metrics. It always inserts the newly found non-dominated schedules into the population, helping to prevent a stagnation

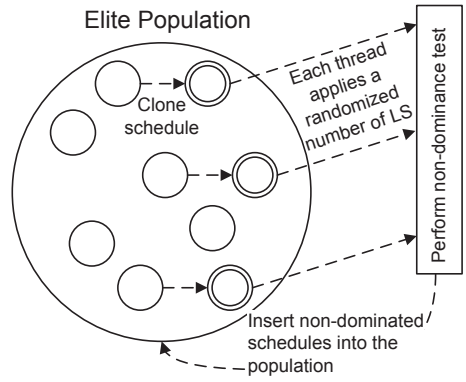

Fig. 1. Diagram of the ME-MLS algorithm

situation. When the population is full, a schedule currently in the population is selected to be replaced based on a distance function, defined as the sum of the relative improvements of each objective metric. Those schedules that compute the minimum value in at least one of the problem objectives are never replaced. When using the fast greedy ad hoc replacement technique in the ME-MLS algorithm we will refer to *ME-MLS + ad hoc* algorithm.

Adaptive Grid Archiving (AGA). The AGA technique was initially proposed as the crowding strategy applied in the Pareto Archived Evolution Strategy (PAES) algorithm [15]. It consists in dividing up the objective space into hypercubes with the goal of balancing the density of non-dominated solutions in the hypercubes. Then, when inserting a non-dominated solution in the Pareto front, its grid location in the solution space is determined. If the Pareto front is already full and the grid location of the new solution does not match with the most crowded hypercube, a solution belonging to that most crowded hypercube is removed before inserting the new one. The AGA strategy guarantees three very desirable properties for multiobjective optimization (MO) algorithms:

1. it maintains solutions at the extremes of all objectives;
2. it maintains solutions in all of the Pareto occupied regions, and
3. it distributes the remaining solutions evenly among the Pareto regions.

When using the AGA replacement technique in the ME-MLS algorithm we will refer to *ME-MLS + AGA* algorithm.

3.3 Local Search

The energy-aware local search performed by each thread in ME-MLS is based on a randomized version of the Problem Aware Local Search (PALS) method [1].

Algorithm 2 Local search applied by each ME-MLS thread on schedule s'

```
1:  search_strategy ← current optimization strategy (makespan, energy, or random)
2:  best_operation ← none
3:  m_A ← random_machine(search_strategy, s')
4:  m_B ← random_machine(search_strategy, s') ≠ m_A
5:  tasks_x ← random_task_set(m_A)
6:  tasks_y ← random_task_set(m_B)
7:  for x ∈ tasks_X do
8:    operator ← random(move or swap)
9:    if operator = swap then
10:     for all y ∈ tasks_Y do
11:       delta_{x,y} ← improvement(search_strategy, s', swap_{x,y})
12:       delta_best ← improvement(search_strategy, s', best_operation)
13:       if delta_{x,y} > delta_best then
14:         best_operation ← swap_{x,y}
15:       end if
16:     end for
17:   else if operator = move then
18:     M ← random_machine_set() ∪{m_B}
19:     for all m_D ∈ M do
20:       delta_{x,m_D} ← improvement(search_strategy, s', move_{x,m_D})
21:       delta_best ← improvement(search_strategy, s', best_operation)
22:       if delta_{x,m_D} > delta_best then
23:         best_operation ← move_{x,m_D}
24:       end if
25:     end for
26:   end if
27: end for
28: apply_operation(best_operation)
```

The local search performed by each thread (Algorithm 2) applies one of the following strategies: *makespan optimization*, *energy optimization*, or *random optimization* (line 1). The method works with two machines m_A and m_B to perform the search (selected in lines 3–4). When the makespan optimization strategy is applied, it selects with high probability the machine defining the makespan as m_A and the machine with the lowest computing time as m_B. The energy optimization strategy selects with high probability the most energy consuming machine as m_A and the less energy consuming machine as m_B. The random optimization selects two random machines as m_A and m_B. Then, a random set of tasks $tasks_X$ from the ones assigned to m_A is selected (line 5). For each task $x \in tasks_X$, the method randomly selects an operation to perform (line 8):

1. a *swap* operation, or
2. a *move* operation.

When the swap operation is applied (lines 9 to 16), the method selects a random set of tasks $tasks_Y$ assigned to m_B (line 6), computes the values for both objectives when swapping task x with each task $y \in tasks_Y$ (lines 11 and 12), and chooses the swap which improves most the schedule objectives (line 14). If the energy and makespan improvements of the computed swap are in conflict (i.e. the swap improves one metric but degrades the other), the best swap according to the selected search strategy is applied. When no swap is found to improve neither of the schedule objectives, no swap is applied.

If the move operation is applied, for task x (lines 17 to 26), the method selects a set of random machines $M = \{randommachines\} \cup \{m_B\}$ (line 18), and computes the variation in the schedule objectives when moving task x to each machine $m_D \in M$ (lines 20–21). Then, the move operation which improves most the schedule objectives prioritizing the currently selected search strategy is selected to be applied (lines 22 to 24).

When no swap/move is found to improve neither of the schedule objectives, no operation is applied.

3.4 Implementation Details

Implementation language and libraries. ME-MLS is implemented in GNU C++ 4.6, avoiding heavy C++ constructs (classes, interfaces, polymorphism) to minimize the code execution overhead.

The multithreading support is provided by the GNU POSIX thread library 2.13. We do not use high level parallel multithreading libraries, such as OpenMP, in order to have fine grain control over the synchronization mechanisms. Finally, the Mersenne Twister (MT) method [23] is used to generate pseudorandom numbers, since empirical evidence indicates that it is more efficient than standard generators [27]. We modified the original MT implementation to be thread-safe, allowing multithreading random numbers generation.

Problem encoding. ME-MLS maintains in memory a multi-structure comprising both a machine-based and a task-based encoding for schedules [25] (Figure 2). This multi-structure allows ME-MLS to efficiently access the tasks assigned to a given machine in $O(1)$ via the machine-based encoding; and given a certain task, to locate the machine to which it is assigned also in $O(1)$ via the task-based encoding.

Population initialization. The population is initialized using a randomized version of the MCT heuristic [22], considering the tasks sorted in a random order. The tasks are scheduled following that random order to the machines that first accomplish them, providing diversity to the initial population.

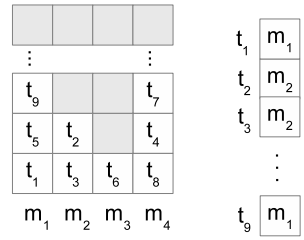

Fig. 2. Problem encodings: a) machine-oriented encoding, b) task-oriented encoding

4 EXPERIMENTAL ANALYSIS

This section introduces the ME-HCSP instances and the hardware platform used in the experimental analysis, as well as the MinMin based heuristics used as a reference baseline to compare the ME-MLS results. Finally, the experimental results are presented and analyzed, along with a speedup evaluation of the proposed method.

4.1 ME-HCSP Instances and Execution Platform

Instances. A set of **96** ME-HCSP instances was generated to evaluate the ME-MLS algorithm. Each instance describes the machines *scenario* and the set of tasks *workloads*. The scenarios were generated using realistic data, gathering for each machine the computational power, and the energy consumption in both idle state and processing state. The workloads were generated following the ETC methodology [2].

Three instance dimensions were used in the experimental analysis (*number of tasks* × *number of machines*): 512 × 16, 1 024 × 32, and 2 048 × 64. For each dimension, 12 different scenarios and 8 workloads (4 using parameters by Ali et al. [2] and 4 using parameters by Braun et al. [4]) were generated. Combining all these scenarios and workloads we obtain a set of 96 ME-HCSP instances that model realistic small- and medium-sized HC infrastructures.

Execution platform. The experimental analysis was performed on a 24-core Magny-Cours AMD Opteron Processor 6172, 2.1 GHz, 24 GB RAM, running 64-bits CentOS 5.1 Linux, from Cluster FING (`http://www.fing.edu.uy/cluster`).

4.2 MinMin Based Heuristics Used as Reference Baseline

In order to compare the ME-MLS results, we propose here a set of heuristics based on the MinMin list scheduling method [22], which is considered to outperform other deterministic heuristics for minimizing the makespan objective [9]. The original MinMin heuristic greedily picks the task that can be completed the soonest, taking

into account the current machine assignments. MinMin starts with a set U of all *unmapped* tasks, calculates the MCT for each task in U for each machine, and assigns the task with the minimum overall MCT to the machine that can complete it faster. The mapped task is removed from U, and the process is repeated until all tasks are mapped. A pseudocode of the MinMin heuristic is presented in Algorithm 3.

Algorithm 3 MinMin heuristic

```
1: U = T {set of unasigned tasks}
2: while U ≠ ∅ do
3:    for each task t_k ∈ U do
4:       for each machine m_h ∈ P do
5:          evaluate ET(t_k, m_h)
6:       end for
7:       store best pair (t_k, m_j)
8:    end for
9:    select pair (t_i, m_j) with minimum ET
10:   assign t_i to m_j
11:   remove t_i from U
12: end while
13: return task assignment
```

MinMin uses a two-phases optimization strategy (the double *for* cycle in Algorithm 3), thus we define four MinMin versions by alternating the minimization objective in each phase. The first version minimizes the completion time in both phases (it corresponds to the classic MinMin heuristic); the second version minimizes the energy objective in both phases; in the remaining versions the minimization objectives are alternated to be in the first phase or in the second phase. We use the lowercase *Min* notation when minimizing the completion time and the uppercase *MIN* notation when minimizing the energy. Thus, the four versions of the MinMin heuristics used in the comparison are: *MinMin*, *MINMIN*, *MinMIN*, and *MINMin*.

4.3 Multiobjective Optimization Metrics Used to Evaluate ME-MLS

Several metrics have been considered in this article to evaluate the two variants of the proposed ME-MLS algorithm, regarding the two main purposes of multiobjective optimization techniques:

1. converging to the Pareto front and
2. correctly sampling the set of non-dominated solutions [5, 6].

The *quality* metrics evaluate the convergence towards the Pareto front. We consider in this work the number of (different) non-dominated solutions found for each algorithm (*ND*), and Inverted Generational Distance (IGD), defined as the (normalized) sum of the distances between the non-dominated solutions in the Pareto front

found by the algorithm and a set of uniformly distributed points in the true Pareto front. Smaller values of IGD mean a better approximation to the Pareto front.

The *diversity* metrics measure the distribution of the computed non-dominated solutions, evaluating the capability of correctly sampling the Pareto front of the problem. In this work we consider the *spread* and *Relative Hypervolume* (RHV) metrics. *Spread* includes the information about the extreme points of the true Pareto front in order to compute a precise value of the distribution [6]. Smaller values of spread mean a better distribution of non-dominated solutions in the calculated Pareto front. RHV is the ratio of the volume (in the objective functions space) covered by the Pareto front computed by the algorithm and the volume covered by the real Pareto front.

In the experimental analysis, the true Pareto front – which is unknown for the ME-HCSP instances studied – was approximated by gathering the non-dominated solutions found using both ME-MLS variants in the 30 independent executions performed for each algorithm.

4.4 Parameter Setting Experiments

The goal of this work is to *efficiently* solve the ME-HCSP, thus a fixed 10 seconds execution time stopping criterion is used for the ME-MLS algorithm. This time stopping criterion is significantly lower than the execution time of the algorithms in the related literature, which ranges from 60s to 90s [28]. For the speedup evaluation of ME-MLS, the stopping criterion was set to 6 million iterations.

In order to provide statistical significance to the results and considering the stochastic nature of the proposed algorithm, 30 independent ME-MLS executions were performed on each instance. Each execution was performed using 24 threads, the maximum number of cores available in the computing platform.

A configuration analysis was performed using the 512×16 dimension instances in order to find the best values for the population size (`POP_SIZE`), the number of local search per schedule (`THREAD_IT`), the neighbourhood size used in the local search (`SRC_TASK_NHOOD`, `DST_TASK_NHOOD`, and `DST_MACH_NHOOD`), and the re-work factor (`REWORK_FACTOR`).

The candidate values for the parameter settings study were: `POP_SIZE` $\in \{30, 35, 40\}$, `THREAD_IT` $\in \{500, 650, 800\}$, `SRC_TASK_NHOOD` $\in \{24, 28, 32\}$, `DST_TASK_NHOOD` $\in \{16, 20, 24\}$, `DST_MACH_NHOOD` $\in \{8, 12, 16\}$, and `REWORK_FACTOR` $\in \{10, 14, 18\}$. The best results were obtained with the following configuration `POP_SIZE` $= 30$, `THREAD_IT` $= 650$, `SRC_TASK_NHOOD` $= 28$, `DST_TASK_NHOOD` $= 16$, `DST_MACH_NHOOD` $= 16$, and `REWORK_FACTOR` $= 14$.

4.5 Results and Discussion

This subsection presents and discusses the experimental results obtained in the evaluation of the ME-MLS scheduling algorithm.

Solution quality. Table 1 reports the average and best makespan and energy improvements by the two studied variants of ME-MLS with respect to the MinMin based heuristic that performs the best for that instance and objective. The results show that ME-MLS outperforms the best MinMin based heuristic with makespan improvements of up to 14.0 % and energy consumption improvements of up to 13.9 %. Both ME-MLS + AGA and ME-MLS + ad hoc are able to compute schedules with similar average objective values for instances with dimension up to 1 024 × 32. ME-MLS + ad hoc computes slightly better average schedules than ME-MLS + AGA when solving instances with dimension 2 048 × 64. In the small ME-HCSP instances (512 × 16), the best improvements are obtained for the makespan objective, while in the largest ME-HCSP instances (2 048 × 64), the best improvements are obtained for the energy consumption. Figure 3 summarizes the average improvements by dimension when comparing ME-MLS to each MinMin based heuristic.

			ME-MLS + AGA				**ME-MLS + ad hoc**			
			makespan		energy		makespan		energy	
dimension 512 × 16	**model**	**heter.**	*best*	*avg*	*best*	*avg*	*best*	*avg*	*best*	*avg*
	Ali	high high	12.9 %	9.4 %	9.5 %	4.4 %	12.3 %	9.1 %	9.7 %	4.4 %
		high low	14.0 **%**	9.5 **%**	9.5 %	5.1 %	13.5 **%**	9.2 **%**	9.5 %	5.2 %
		low high	10.9 %	8.3 %	9.3 %	5.3 %	10.9 %	8.4 %	9.2 %	5.2 %
		low low	10.9 %	4.1 %	10.7 **%**	7.4 **%**	11.2 %	4.0 %	10.7 **%**	7.4 **%**
	Braun	high high	11.7 %	9.5 %	9.7 %	6.3 %	11.6 %	9.2 %	10.0 %	6.2 %
		high low	5.5 %	3.8 %	5.3 %	2.3 %	5.4 %	3.8 %	5.2 %	2.3 %
		low high	10.3 %	8.0 %	8.3 %	4.1 %	10.5 %	7.8 %	8.3 %	4.2 %
		low low	5.9 %	5.0 %	5.5 %	3.2 %	5.9 %	5.0 %	4.7 %	3.1 %
			makespan		energy		makespan		energy	
dimension 1 024 × 32	**model**	**heter.**	*best*	*avg*	*best*	*avg*	*best*	*avg*	*best*	*avg*
	Ali	high high	9.7 %	7.3 %	9.6 %	5.9 %	10.5 %	7.1 %	10.0 %	6.2 %
		high low	8.7 %	7.2 %	8.9 %	5.6 %	9.7 %	6.7 %	9.2 %	5.4 %
		low high	10.8 **%**	8.7 %	9.9 %	7.8 %	11.3 %	9.0 %	9.8 %	7.3 %
		low low	6.8 %	2.3 %	11.4 **%**	8.9 **%**	7.0 %	2.6 %	11.4 **%**	8.8 **%**
	Braun	high high	8.8 %	6.2 %	9.3 %	5.7 %	9.1 %	6.1 %	10.3 %	6.3 %
		high low	10.1 %	7.6 %	9.4 %	6.0 %	10.3 %	7.5 %	9.8 %	6.3 %
		low high	8.5 %	6.0 %	7.9 %	4.4 %	8.7 %	5.8 %	8.0 %	4.7 %
		low low	10.5 %	9.0 **%**	11.3 %	8.8 %	11.3 **%**	9.3 **%**	11.0 %	8.5 %
			makespan		energy		makespan		energy	
dimension 2 048 × 64	**model**	**heter.**	*best*	*avg*	*best*	*avg*	*best*	*avg*	*best*	*avg*
	Ali	high high	7.2 %	5.0 %	9.4 %	4.6 %	8.5 %	5.7 %	11.1 %	6.8 %
		high low	6.2 %	4.2 %	9.2 %	4.6 %	7.7 %	5.2 %	10.7 %	6.5 %
		low high	6.8 %	4.5 %	12.7 %	9.9 %	7.6 %	6.3 %	12.7 %	9.4 %
		low low	3.4 %	0.3 %	12.9 %	9.3 %	4.8 %	1.1 %	13.3 %	9.2 %
	Braun	high high	7.2 %	4.7 %	9.9 %	5.3 %	8.1 %	5.4 %	10.8 %	7.3 %
		high low	7.6 **%**	6.2 **%**	9.5 %	6.6 %	7.8 %	6.4 **%**	10.4 %	8.3 %
		low high	7.4 %	5.0 %	9.9 %	4.8 %	8.8 **%**	5.6 %	11.7 %	7.0 %
		low low	6.4 %	4.3 %	13.9 **%**	10.1 **%**	8.0 %	6.0 %	13.8 **%**	9.6 **%**

Table 1. ME-MLS improvements over the best MinMin heuristic

Multiobjective optimization metrics. We compared ME-MLS + AGA and ME-MLS + ad hoc by studying the multiobjective optimization metrics on the Pareto front approximations computed by each algorithm. In order to determine the significance of the comparison, a statistical analysis was performed

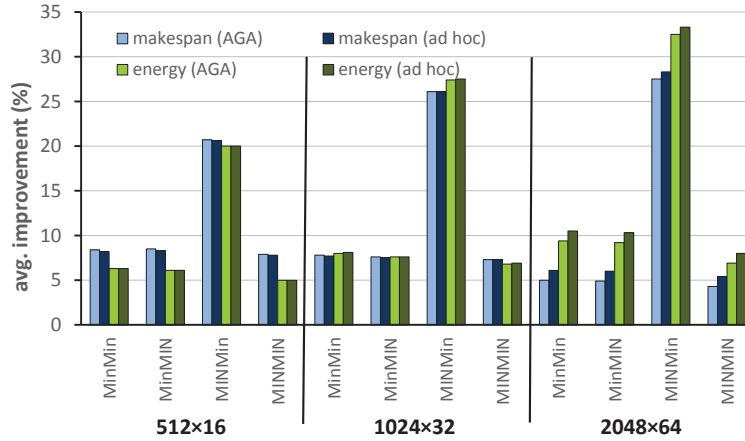

Fig. 3. Average ME-MLS improvements over the MinMin based heuristics

over the results for each algorithm, metric, and problem instance solved. First, the Kolmogorov-Smirnov (K-S) test was applied to check whether the metric values follow a normal distribution or not. The values for the D statistic by the K-S test indicated that the results for ME-MLS + AGA and ME-MLS + ad hoc are not normally distributed. As a consequence, the non-parametric Kruskal-Wallis statistical test was performed with a confidence level of 95 %, to compare the distributions for ME-MLS + AGA and ME-MLS + ad hoc.

Table 2 reports the average and standard deviation values for the ND and IGD quality metrics for each instance model, heterogeneity class, and problem dimension.

Table 3 reports the average and standard deviation values for spread and RHV diversity metrics for each instance model, heterogeneity class, and problem dimension. For each metric and heterogeneity class, the best algorithm and the number of problem instances in which it is the best with 95 % confidence (i.e. the computed pairwise p-value is below 5×10^{-2}) is also reported in the $\text{best}_{95\,\%}$ column. The result is emphasized in **bold font** when a given algorithm variant is always better than the other one in the twelve problem instances solved for each instance model, heterogeneity class, and problem dimension.

Table 2 demonstrates that ME-MLS + AGA outperforms ME-MLS + ad hoc in terms of the quality metrics, specially in the largest ME-HCSP instances. Regarding the diversity metrics, Table 3 shows that both algorithms compute similar results for spread and RHV for dimension 512×16, but ME-MLS + AGA was able to find Pareto fronts with better diversity and covering properties for dimension $1\,024 \times 32$ and $2\,048 \times 64$, as it is demonstrated by the values of the spread and RHV metrics.

	model	heter.	ND			IGD (normalized)		
			AGA	ad hoc	best$_{95\,\%}$	AGA	ad hoc	best$_{95\,\%}$
dimension 512 × 16	Ali	high high	8.33 ± 2.47	4.11 ± 1.43	AGA 10/12	1.00 ± 0.39	1.57 ± 0.76	AGA 6/12
		high low	8.02 ± 2.65	4.20 ± 1.39	AGA 8/12	1.00 ± 0.44	1.51 ± 0.73	AGA 5/12
		low high	6.67 ± 2.42	3.82 ± 1.20	AGA 10/12	1.00 ± 0.34	1.56 ± 0.64	AGA 8/12
		low low	2.84 ± 1.54	2.49 ± 1.12	ad hoc 3/12	1.00 ± 0.40	1.04 ± 0.40	ad hoc 1/12
	Braun	high high	6.63 ± 3.06	3.75 ± 1.50	AGA 7/12	1.00 ± 0.43	1.53 ± 0.64	AGA 5/12
		high low	6.18 ± 3.30	3.36 ± 1.50	AGA 8/12	1.00 ± 0.33	1.65 ± 0.67	AGA 6/12
		low high	6.64 ± 3.03	3.84 ± 1.56	AGA 7/12	1.00 ± 0.45	1.31 ± 0.61	AGA 5/12
		low low	6.02 ± 3.10	3.21 ± 1.21	AGA 7/12	1.00 ± 0.36	1.59 ± 0.63	AGA 7/12
dimension 1 024 × 32			AGA	ad hoc	best$_{95\,\%}$	AGA	ad hoc	best$_{95\,\%}$
	Ali	high high	9.98 ± 0.22	4.44 ± 1.31	AGA 12/12	1.00 ± 0.32	2.38 ± 0.68	AGA 12/12
		high low	9.99 ± 0.09	4.16 ± 1.38	AGA 12/12	1.00 ± 0.32	2.62 ± 0.87	AGA 10/12
		low high	8.67 ± 1.48	3.63 ± 1.01	AGA 12/12	1.00 ± 0.23	3.35 ± 1.01	AGA 12/12
		low low	5.90 ± 1.90	2.92 ± 0.84	AGA 10/12	1.00 ± 0.43	1.57 ± 0.62	AGA 5/12
	Braun	high high	9.98 ± 0.16	4.14 ± 1.23	AGA 12/12	1.00 ± 0.28	2.29 ± 0.63	AGA 12/12
		high low	9.95 ± 0.43	4.72 ± 1.35	AGA 12/12	1.00 ± 0.29	2.48 ± 0.80	AGA 10/12
		low high	9.96 ± 0.42	4.75 ± 1.30	AGA 12/12	1.00 ± 0.32	2.38 ± 0.83	AGA 12/12
		low low	8.56 ± 1.49	3.63 ± 0.99	AGA 12/12	1.00 ± 0.24	3.06 ± 0.97	AGA 12/12
dimension 2 048 × 64			AGA	ad hoc	best$_{95\,\%}$	AGA	ad hoc	best$_{95\,\%}$
	Ali	high high	10.00 ± 0.00	5.25 ± 0.92	AGA 12/12	1.00 ± 0.26	2.04 ± 0.62	AGA 12/12
		high low	10.00 ± 0.05	5.20 ± 0.99	AGA 12/12	1.00 ± 0.25	2.13 ± 0.67	AGA 10/12
		low high	9.09 ± 1.19	3.72 ± 1.10	AGA 12/12	1.00 ± 0.20	3.19 ± 0.92	AGA 12/12
		low low	8.02 ± 1.63	3.66 ± 1.18	AGA 12/12	1.00 ± 0.26	1.47 ± 0.53	AGA 6/12
	Braun	high high	10.00 ± 0.00	5.19 ± 0.93	AGA 12/12	1.00 ± 0.26	2.13 ± 0.65	AGA 12/12
		high low	10.00 ± 0.00	5.18 ± 0.92	AGA 12/12	1.00 ± 0.26	2.19 ± 0.77	AGA 12/12
		low high	10.00 ± 0.00	5.23 ± 0.87	AGA 12/12	1.00 ± 0.29	2.00 ± 0.65	AGA 12/12
		low low	9.49 ± 0.86	4.03 ± 1.16	AGA 12/12	1.00 ± 0.19	2.92 ± 1.00	AGA 12/12

Table 2. ME-MLS multiobjective quality metrics

The sample Pareto front approximations in Figure 4 show that the ME-MLS + AGA algorithm is able to compute accurate solutions, with good diversity and an appropriate covering.

Summarizing, the study indicates that the AGA replacement strategy is a useful choice to improve the quality and diversity of the schedules found by the ME-MLS algorithm, significantly improving over the ad hoc replacement technique regarding the two main goals in MO.

Computational efficiency: speedup evaluation. A speedup analysis was performed for both ME-MLS + AGA and ME-MLS + ad hoc algorithms in order to study the reductions in the execution times when using different numbers of threads. The speedup evaluation was done over the 1 024 × 32 instances, and performing 30 independent executions of each version of the algorithm, using a stopping criterion of 6 million iterations.

Figure 5 graphically summarizes the results of the speedup analysis when using up to 24 threads executing on the 24 cores available in the Magny-Cours experimental computing platform used.

The study indicates that both ME-MLS + AGA and ME-MLS + ad hoc algorithms have a near linear speedup behavior. ME-MLS + AGA is the most efficient and scalable algorithm: it achieves a speedup of 22.4 when using 24 threads, while ME-MLS + ad hoc obtains a speedup of 20.2.

	model	heter.	spread			RHV		
			AGA	ad hoc	best$_{95\,\%}$	AGA	ad hoc	best$_{95\,\%}$
dimension 512 × 16	Ali	high high	1.00 ± 0.12	1.07 ± 0.28	AGA 5/12	0.85 ± 0.06	0.79 ± 0.07	AGA 6/12
		high low	1.00 ± 0.10	1.05 ± 0.26	ad hoc 4/12	0.82 ± 0.08	0.78 ± 0.08	AGA 6/12
		low high	1.00 ± 0.15	1.20 ± 0.32	AGA 6/12	0.89 ± 0.05	0.85 ± 0.06	AGA 8/12
		low low	1.01 ± 0.15	1.00 ± 0.17	ad hoc 2/12	0.82 ± 0.09	0.81 ± 0.08	none
	Braun	high high	1.00 ± 0.13	1.02 ± 0.29	none	0.84 ± 0.07	0.79 ± 0.08	AGA 7/12
		high low	1.00 ± 0.14	1.08 ± 0.33	AGA 5/12	0.91 ± 0.04	0.88 ± 0.04	AGA 6/12
		low high	1.00 ± 0.11	1.05 ± 0.25	AGA 3/12	0.80 ± 0.09	0.77 ± 0.09	AGA 6/12
		low low	1.00 ± 0.14	1.12 ± 0.30	AGA 7/12	0.92 ± 0.03	0.91 ± 0.03	AGA 5/12
			AGA	ad hoc	best$_{95\,\%}$	AGA	ad hoc	best$_{95\,\%}$
dimension 1 024 × 32	Ali	high high	1.00 ± 0.15	1.30 ± 0.40	AGA 9/12	0.86 ± 0.05	0.75 ± 0.07	AGA 11/12
		high low	1.00 ± 0.13	1.17 ± 0.39	AGA 7/12	0.84 ± 0.05	0.72 ± 0.08	AGA 11/12
		low high	1.00 ± 0.15	1.39 ± 0.46	AGA 10/12	0.92 ± 0.03	0.81 ± 0.05	AGA 11/12
		low low	1.00 ± 0.17	1.00 ± 0.34	AGA 2/12	0.90 ± 0.04	0.89 ± 0.04	AGA 5/12
	Braun	high high	1.00 ± 0.15	1.21 ± 0.42	AGA 7/12	0.86 ± 0.05	0.77 ± 0.06	AGA 11/12
		high low	1.00 ± 0.13	1.32 ± 0.36	AGA 8/12	0.85 ± 0.05	0.74 ± 0.07	AGA 11/12
		low high	1.00 ± 0.15	1.37 ± 0.39	AGA 10/12	0.86 ± 0.05	0.76 ± 0.06	AGA 11/12
		low low	1.00 ± 0.14	1.40 ± 0.43	AGA 10/12	0.91 ± 0.03	0.82 ± 0.05	AGA 11/12
			AGA	ad hoc	best$_{95\,\%}$	AGA	ad hoc	best$_{95\,\%}$
dimension 2 048 × 64	Ali	high high	1.00 ± 0.14	1.47 ± 0.37	AGA 10/12	0.82 ± 0.04	0.76 ± 0.08	AGA 9/12
		high low	1.00 ± 0.14	1.45 ± 0.38	AGA 10/12	0.83 ± 0.04	0.77 ± 0.07	AGA 8/12
		low high	1.00 ± 0.15	1.35 ± 0.43	AGA 10/12	0.88 ± 0.03	0.81 ± 0.06	AGA 11/12
		low low	1.00 ± 0.14	1.17 ± 0.36	AGA 5/12	0.82 ± 0.06	0.84 ± 0.04	ad hoc 6/12
	Braun	high high	1.00 ± 0.14	1.46 ± 0.37	AGA 10/12	0.83 ± 0.04	0.76 ± 0.07	AGA 11/12
		high low	1.00 ± 0.13	1.44 ± 0.33	AGA 8/12	0.86 ± 0.03	0.78 ± 0.07	AGA 11/12
		low high	1.00 ± 0.15	1.49 ± 0.34	AGA 10/12	0.83 ± 0.04	0.76 ± 0.07	AGA 10/12
		low low	1.00 ± 0.15	1.43 ± 0.37	AGA 10/12	0.88 ± 0.02	0.81 ± 0.05	AGA 11/12

Table 3. ME-MLS multiobjective diversity metrics

5 CONCLUSIONS

In this article, we have proposed, implemented, and evaluated a new fast parallel multiobjective local search algorithm to tackle the energy-aware scheduling problem in HC systems. The new scheduler, called ME-MLS, is based on Pareto dominance, and it works on a population of non-dominated schedules that are iteratively evolved in parallel by using multithreading programming techniques. In order to demonstrate the good performance of the algorithm, an efficient termination condition of only 10 seconds was set in the experimental evaluation, allowing to perform almost-online scheduling in HC systems.

The experimental analysis compared the proposed method versus state-of-the-art scheduling techniques. Two different versions of the multiobjective scheduling ME-MLS algorithm were evaluated, differing on the policy followed to discard non-dominated solutions when the population is full: the ME-MLS + AGA and the ME-MLS + ad hoc algorithms.

The two proposed versions of the algorithm were evaluated on a testbed of 96 problem instances with dimensions of up to 2 048 tasks and 64 processors, and they were validated by comparing the obtained results versus those of four versions of the state-of-the-art MinMin scheduler: the classical one and three other energy-efficient ones. The Kruskal-Wallis non-parametric test was applied to look

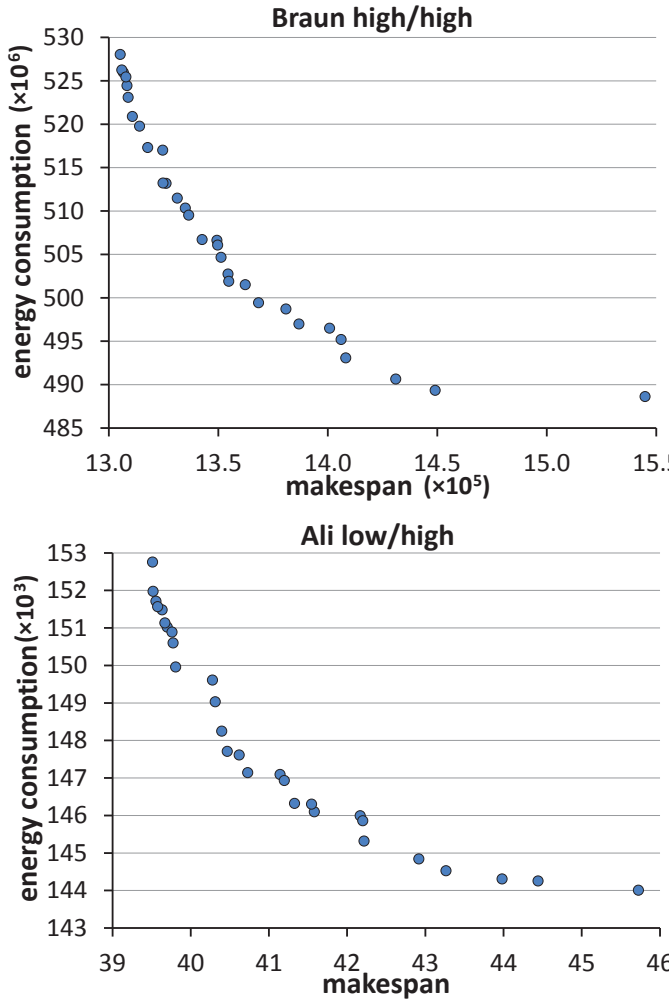

Fig. 4. Approximated Pareto fronts computed by the ME-MLS + AGA algorithm

for statistical significance on the algorithms comparison, and we concluded that ME-MLS + AGA offers a better overall performance than ME-MLS + ad hoc.

The experimental analysis shows that ME-MLS + AGA outperforms best Min-Min based heuristic for every instance, with improvements of up to 14.0 % for the makespan and up to 13.9 % in the energy consumption. The performance analysis shows that the two ME-MLS variants have a promising scalability behavior.

As future work, we are working on improving ME-MLS to make it quicker and more accurate, by designing new specialized operators for the problem. Additionally, we plan to integrate the proposed local search algorithms into well-known and

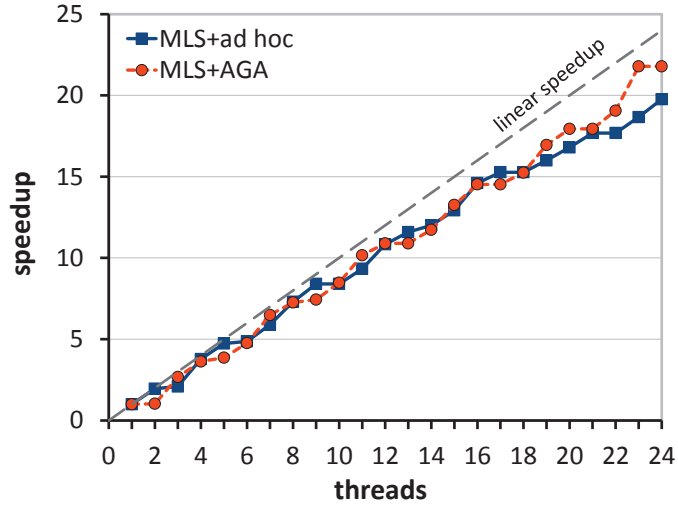

Fig. 5. Speedup for the two variants of the ME-MLS algorithm

accurate multiobjective EAs to look for higher quality solutions. We also plan to tackle dynamic versions of the scheduling problem in HC/grid environments.

Acknowledgments

The work of S. Nesmachnow and S. Iturriaga is partly supported by Agencia Nacional de Investigación e Innovación (ANII) and Programa de Desarrollo de las Ciencias Básicas (PEDECIBA), Uruguay. B. Dorronsoro acknowledges that he present work is partially supported by the National Research Fund, Luxembourg, and cofunded under the Marie Curie Actions of the European Commission (FP7-COFUND). P. Bouvry would like to acknowledge that this work was partly supported by Luxembourg FNR GreenIT project (C09/IS/05).

REFERENCES

[1] Alba, E.—Luque, G.: A new Local Search Algorithm for the DNA Fragment Assembly Problem. In Proc. of 7th European Conf. on Evolutionary Computation in Combinatorial Optimization 2007, pp. 1–12.

[2] Ali, S.—Siegel, H.—Maheswaran, M.—Ali, S.—Hensgen, D.: Task Execution Time Modeling for Heterogeneous Computing Systems. In Proc. of 9th Heterogeneous Computing Workshop 2000, pp. 185–192.

[3] Berman, F.—Fox, G.—Hey, A.: Grid Computing: Making the Global Infrastructure a Reality. Wiley 2003.

[4] Braun, T.—Siegel, H.—Beck, N.—Bölöni, L.—Maheswaran, M.—Reuther, A.—Robertson, J.—Theys, M.—Yao, B.—Hensgen, D.—Freund, R.: A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks into Heterogeneous Distributed Computing Systems. J. Parallel Distrib. Comput., Vol. 61, 2001, No. 6, pp. 810–837.

[5] Coello, C.—Lamont, G.—van Veldhuizen, D.: Evolutionary Algorithms for Solving Multi-Objective Problems. Springer, Berlin, Heidelberg 2007.

[6] Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. Wiley-Interscience Series in Systems and Optimization, John Wiley & Sons, Chichester 2001.

[7] Foster, I.—Kesselman, C.: The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann 1998.

[8] Garey, M.—Johnson, D.: Computers and intractability. Freeman 1979.

[9] Izakian, H.—Abraham, A.—Snasel, V.: Comparison of Heuristics for Scheduling Independent Tasks on Heterogeneous Distributed Environments. In Proc. of Int. Joint Conf. on Comp. Sciences and Optimization, Washington, DC, USA 2009, pp. 8–12.

[10] Kessaci, Y.—Mezmaz, M.—Melab, N.—Talbi, E. G.—Tuyttens, D.: Parallel Evolutionary Algorithms for Energy Aware Scheduling. Intelligent Decision Systems in Large-Scale Distributed Environments, Springer 2011.

[11] Khan, S.—Ahmad, I.: A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids. IEEE Trans. Parallel Distrib. Syst., Vol. 20, 2009, pp. 346–360.

[12] Kim, J.-K.—iegel, H.—Maciejewski, A.—Eigenmann, R.: Dynamic Resource Management in Energy Constrained Heterogeneous Computing Systems Using Voltage Scaling. IEEE Trans. Parallel Distrib. Syst., Vol. 19, 2008, pp. 1445–1457.

[13] Kim, K.—Buyya, R.—Kim, J.: Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-Enabled Clusters. In Proc. of 7th IEEE Int. Symposium on Cluster Computing and the Grid 2007, pp. 541–548.

[14] Knowles, J.—Corne, D.: Properties of an Adaptive Archiving Algorithm for Storing Nondominated Vectors. IEEE Trans. Evol. Comput., Vol. 7, 2003, No. 2, pp. 100–116.

[15] Knowles, J.—Corne, D.: Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. Evol. Comput., Vol. 8, 2000, No. 2, pp. 149–172.

[16] Kolodziej, J.—Khan, S.—Wang, L.—Byrski, A.—Min-Allah, N.—Madani, S.: Hierarchical genetic-based grid scheduling with energy optimization. Cluster Computing, DOI:10.1007/s10586–012-0226–7, 2012.

[17] Lee, Y.-C.—Zomaya, A.: Minimizing Energy Consumption for Precedence Constrained Applications Using Dynamic Voltage Scaling. In Proc. of 9th IEEE/ACM Int. Symposium on Cluster Computing and the Grid 2009, pp. 92–99.

[18] Lee, Y.-C.—Zomaya, A.: Energy Conscious Scheduling for Distributed Computing Systems Under Different Operating Conditions. IEEE Trans. Parallel Distrib. Syst., Vol. 22, 2011, pp. 1374–1381.

[19] Leung, J.—Kelly, L.—Anderson, J.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press 2004.

[20] Li, Y.—Liu, Y.—Qian, D.: A Heuristic Energy-Aware Scheduling Algorithm for Heterogeneous Clusters. In Proc. of 15th Int. Conf. on Parallel and Distributed Systems 2009, pp. 407–413.

[21] Lindberg, P.—Leingang, J.—Lysaker, D.—Khan, S.—Li, J.: Comparison and Analysis of Eight Scheduling Heuristics for the Optimization of Energy Consumption and Makespan in Large-Scale Distributed Systems. Journal of Supercomputing, Vol. 59, 2012, No. 1, pp. 323–360.

[22] Luo, P.—Lu, K.—Shi, Z.: A Revisit of Fast Greedy Heuristics for Mapping a Class of Independent Tasks Onto Heterogeneous Computing Systems. J. Parallel Distrib. Comput., Vol. 67, 2007, No. 6, pp. 695–714.

[23] Matsumoto, M.—Nishimura, T.: Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. ACM Trans. Model. Comput. Simul., Vol. 8, 1998, No. 1, pp. 3–30.

[24] Mezmaz, M.—Melab, N.—Kessaci, Y.—Lee, Y.—Talbi, E. G.—Zomaya, A.—Tuyttens, D.: A Parallel Bi-Objective Hybrid Metaheuristic for Energy-Aware Scheduling for Cloud Computing Systems. J. Parallel Distrib. Comput., Vol. 71, 2011, No. 11, pp. 1497–1508.

[25] Nesmachnow, S.—Cancela, H.—Alba, E.: Heterogeneous Computing Scheduling with Evolutionary Algorithms. Soft Computing, Vol. 15, 2010, No. 4, pp. 685–698.

[26] Nesmachnow, S.—Iturriaga, S.: Multiobjective Scheduling on Distributed Heterogeneous Computing and Grid Environments Using a Parallel Micro-CHC Evolutionary Algorithm. In Proc. of Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing 2011, pp. 134–141.

[27] Nesmachnow, S.—Luna, F.—Alba, E.: Time Analysis of Standard Evolutionary Algorithms as Software Programs. In Proc. of 11th Int. Conf. on Intelligent Systems Design and Applications 2011, pp. 271–276.

[28] Nesmachnow, S.—Cancela, H.—Alba, E.: A Parallel Micro Evolutionary Algorithm for Heterogeneous Computing and Grid Scheduling. Appl. Soft Comput., Vol. 12, 2012, No. 2, pp. 626–639.

[29] Pecero, J.—Bouvry, P.—Fraire, H.—Khan, S.: A Multi-Objective GRASP Algorithm for Joint Optimization of Energy Consumption and Schedule Length of Precedence-Constrained Applications. In Proc. of Int. Conf. Cloud and Green Computing 2011, pp. 1–8.

[30] Pinel, F.—Dorronsoro, B.—Bouvry, P.: Solving Very Large Instances of the Scheduling of Independent Tasks Problem on the GPU. Journal of Parallel and Distributed Computing 2012 (to appear).

[31] Pinel, F.—Dorronsoro, B.—Pecero, J.—Bouvry, P.—Khan, S.: A Two Phase Heuristic for the Energy-Efficient Scheduling of Independent Tasks on Computational Grids. Journal of Cluster Computing 2012 (to appear).

[32] Xhafa, F.—Duran, B.: Parallel Memetic Algorithms for Independent job Scheduling in Computational Grids. Recent Advances in Evolutionary Computation for Combinatorial Optimization, Springer 2008, pp. 219–239.

[33] Zhu, D.—Melhem, R.—Childers, B.: Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems. IEEE Trans. Parallel Distrib. Syst., Vol. 14, 2003, pp. 686–700.

Santiago Iturriaga has a degree in engineering and a M. Sc. in computer science from Universidad de la República, Uruguay. He is an assistant at Engineering Faculty, Universidad de la República. His M. Sc. thesis is focused on high performance metaheuristics for scheduling. He has published journal and conference articles in these areas.

Sergio Nesmachnow has a degree in engineering, a M. Sc. and a Ph. D. in computer science from Universidad de la República, Uruguay. He is currently a Full Professor at Numerical Computing Center, Engineering Faculty, Universidad de la República, working on scientific high performance computing, parallel metaheuristics, and their application for solving complex real-world problems. He has published over 50 papers in international journals and conference proceedings, has been member of technical program committees of many international conferences, and serves as editor and reviewer for leading journals and conferences.

Bernabé Dorronsoro has a degree in engineering and a Ph. D. in computer science from University of Málaga, Spain, and he is currently working as research associate at University of Luxembourg. His research interests include grid computing, ad hoc networks, the design of efficient meta-heuristics, and their application for solving complex real-world problems in logistics, telecommunications, bioinformatics, combinatorial, multiobjective, and global optimization. He has several articles in impact journals and one book, has been a program committee memberof several conferences and workshops, and he usually serves as reviewer for leading journals and conferences.

Pascal Bouvry has a Ph. D. in computer science at University of Grenoble, France. He is now Professor at the Faculty of Sciences, Technology and Communication, University of Luxembourg and head of the Computer Science and Communication research unit (`http://csc.uni.lu`). He is specialized in parallel and evolutionary computing and his current interest concerns the application of nature-inspired computing for solving reliability, security, and energy-efficiency problems in clouds, grids and ad-hoc networks.