J Grid Computing (2013) 11:653–680
DOI 10.1007/s10723-013-9258-3

Energy-Aware Scheduling on Multicore Heterogeneous Grid Computing Systems

Sergio Nesmachnow · Bernabé Dorronsoro · Johnatan E. Pecero · Pascal Bouvry

Received: 26 October 2012 / Accepted: 6 May 2013 / Published online: 24 May 2013

Abstract We address a multicriteria non-preemptive energy-aware scheduling problem for computational Grid systems. This work introduces a new formulation of the scheduling problem for multicore heterogeneous computational Grid systems in which the minimization of the energy consumption, along with the makespan metric, is considered. We adopt a two-level model, in which a meta-broker agent (level 1) receives all user tasks and schedules them on the available resources, belonging to different local providers (level 2). The computing capacity and energy consumption of resources are taken from real multi-core processors from the main current vendors. Twenty novel list scheduling methods for the problem are proposed, and a comparative analysis of all of them over a large set of problem instances is presented. Additionally, a scalability study is performed in order to analyze the contribution of the best new bi-objective list scheduling heuristics when the problem dimension grows. We conclude after the experimental analysis that accurate trade-off schedules are computed by using the new proposed methods.

Keywords Grid scheduling · Energy-aware · Heterogeneous computing

S. Nesmachnow (✉)
Universidad de la República, Montevideo, Uruguay
e-mail: sergion@fing.edu.uy

B. Dorronsoro
LIFL, University of Lille 1,
Villeneuve d'Ascq Cedex, France
e-mail: bernabe.dorronsoro_diaz@inria.fr

J. E. Pecero · P. Bouvry
University of Luxembourg,
Luxembourg City, Luxembourg

J. E. Pecero
e-mail: johnatan.pecero@uni.lu

P. Bouvry
e-mail: pascal.bouvry@uni.lu

1 Introduction

Nowadays, computing platforms usually comprise a heterogeneous collection of computers. Heterogeneous Computing (HC) systems have been increasingly employed to solve complex problems in the last twenty years, mainly due to the fast increase of the processing power of low-cost computers and the rapid development of high-speed networking technologies. Distributed computing techniques have been applied both in cluster and

Grid computing systems, conceived as a large loosely-coupled virtual supercomputer formed by combining many heterogeneous platforms of different characteristics. Grid infrastructures have made it feasible to provide pervasive and cost-effective access to distributed computing resources for solving hard problems [23].

A key problem when using distributed HC Grid systems consists in finding an optimal scheduling strategy for a set of tasks to be executed [17, 59]. In the scheduling problem, the goal is to assign tasks to the computing resources by reflecting some user- and system-centric objectives and satisfying some efficiency criteria, usually related to the total execution time, resource utilization, economic cost, etc. Scheduling problems on homogeneous computing systems have been widely studied in operational research [20, 46]. However, the *heterogeneous computing scheduling problem* (HCSP) became important due to the popularization of distributed computing and the growing use of heterogeneous clusters and Grid systems since the late 1990's [21, 25].

Traditional scheduling problems are NP-hard [26]. Thus, classic exact and enumerative methods are only useful to solve reduced size problem instances. Grid scheduling is a very challenging problem, as it is crucial to maximize the use of the computing resources. Practitioners in Grid scheduling are searching for suitable implementations of effective and efficient algorithms [14, 47, 50, 69]. In this context, heuristics are useful methods to solve the HCSP, since they are able to compute accurate sub-optimal schedules in reasonable time.

Many scheduling heuristics have been proposed to minimize the total execution time for a set of independent tasks [51]. However, the scheduling problem is inherently multi-objective, especially in the Grid scenario, because it usually implies several conflicting objectives to optimize, such as those matching the interests of the service provider (i.e., *makespan*, cost, etc.) and the customer (especially those related to the QoS) [39, 64, 69, 70]. The simultaneous optimization of several objective functions has been usually tackled by using aggregative functions in the related literature. The drawback of this approach is that the search, and therefore the obtained results, are strongly biased by the parameters chosen for the weight vector. Moreover, from the combinatorial perspective, a number of efficient solutions, known as non-supported efficient solutions, are not optimal for any weighted-sum aggregation function. Additionally, classical multi-objective metaheuristics require longer execution times and report a large number of non-dominated solutions [19, 72], needing the intervention of a decision maker to choose one among them.

Recently, power provisioning and energy consumption have become great challenges in the field of high performance computing. Energy consumption is now of major interest, raising various monetary, environmental, and system performance concerns [43]. Energy consumption on many computing systems is not only dependent on the energy efficiency of the hardware, but also on the software deployed on the infrastructure to manage the resources, especially the resource management system or the scheduling algorithms. Thus, researchers have focused on developing energy-aware scheduling algorithms for HC systems. Providers will obviously be always interested in more energy-efficient solutions with the same QoS level. Most of the proposed algorithms target the energy optimization either in a single machine or at the node level, showing to be effective. However, energy efficiency at the Grid level is of recent interest. The main trend is to optimize the energy consumption of the computing elements, because the processor is the main consumer of energy, and frequently it also offers the most flexible options for energy management, such as dynamic voltage scaling (DVS), dynamic power management, slack sharing and reclamation, and other techniques [35, 37, 43, 58, 73].

In this work, we target the design of twenty highly efficient multi-objective heuristics that provide different trade-off solutions to the multi-objective scheduling problem of independent tasks in HC Grid systems, minimizing both the makespan and the energy consumption. These heuristics are modifications and combinations in different ways of state-of-the-art simple heuristics for the makespan optimization of the independent tasks scheduling problem. In contrast to the

existing multi-objective techniques, these novel heuristics are suitable for being implemented in a real system, because they provide one single accurate trade-off solution in a quick response, overcoming the previously commented drawbacks of aggregative and Pareto approaches.

The proposed heuristics search in distinct areas of the Pareto front, providing minimal energy consumption solutions for different makespan levels. Then, the system designer might use the most appropriate heuristic among the presented ones for his specific case. In this work, the different compromise solutions provided by the heuristics are analyzed and compared, and the best overall heuristic according to the two objectives, as well as its accuracy when scaling the problem in the number of tasks and/or machines, is identified.

In the problem model we define, tasks are supposed not to be subject to deadline or precedence constraints, and a given schedule is evaluated by explicitly computing both the makespan and energy consumption objectives. We address offline scheduling under consideration of zero release times (without loss of generality) for the independent tasks or applications. While real Grid systems exhibit online behavior, it is known that tasks typically remain in queues for a significant time. Therefore, an offline scheduling strategy is beneficial for the efficient online scheduling of such a set of tasks in the current queue. Moreover, many offline scheduling algorithms exhibit a good performance also in the online case. Based on theoretical results, it is known that the performance bounds of offline scheduling strategies can be approximated for the online cases [32, 65].

Our approach does not apply DVS nor other techniques as energy-conscious scheduling [43, 44, 57], relative superiority [43, 52], or distributed power management [40]. Instead, we propose twenty fast methods that can be classified into the class of *list scheduling heuristics* [41], adapted to take into account both makespan and energy consumption in the search for optimal schedules for HC Grid systems composed by multicore computers. Therefore, our heuristics perform a two dimensions energy optimization: within a computer (of multiple cores) and across different Grid nodes.

The main contributions of the research reported in this article are: i) the formulation of a novel mathematical model for the energy consumption in multicore HC Grid systems; ii) the definition of a new multi-objective scheduling problem for HC Grid systems to minimize makespan and energy consumption; iii) the design and evaluation of twenty new fast bi-objective list scheduling heuristics for the problem; iv) the generation and resolution of new large problem instances; and v) a scalability analysis studying the efficacy of the best heuristics found when facing scenarios with different ratios between the number of tasks and machines.

The manuscript is organized as follows. Section 2 presents the Grid system model and the bi-objective scheduling problem formulation. The model for computing the energy consumption for a given schedule in a multicore HC Grid system is introduced in Section 3. Section 4 reviews the related work in the addressed topics, including a description of the class of list scheduling techniques and energy optimization in Grid scheduling. The twenty bi-objective heuristics proposed in this work are introduced in Section 5. After that, the experimental evaluation of the proposed heuristics over a large set of problem instances is described in Section 6. Finally, Section 7 presents the conclusions of the research and formulates the main lines for future work.

2 Heterogeneous Computing Scheduling

We consider a two-level Grid system [32, 39, 59, 60, 64, 69], based on the Meta-Broker model [27]. In this model, there is a meta-scheduler at the highest level of the hierarchy, and a number of local brokers at the second level (see a diagram in Fig. 1).

Local brokers collect information about the computing capabilities of the Grid resources and send it to the meta-scheduler, which also receives applications from Grid users to be processed in a batch mode. With this information, the meta-broker selects the most appropriate set of resources to for the tasks and generates optimized

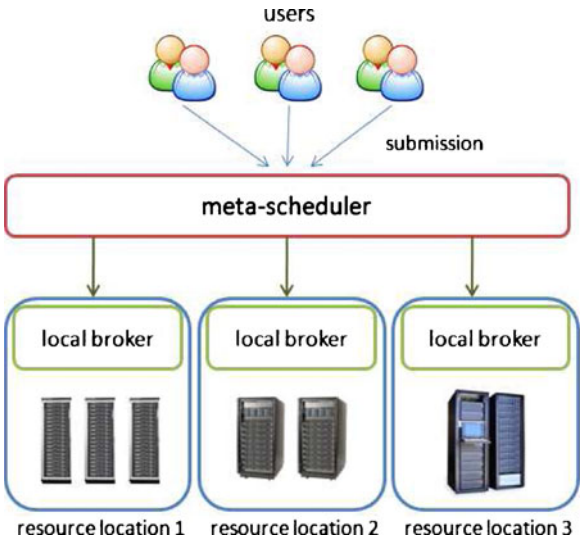


Fig. 1 Scheduling in a hierarchical Grid system

schedules according to some criteria, both targeting system and user interests [27, 38, 63].

In this work, we only consider the optimization at a single Grid system. However, some researchers consider an interoperable Grid system or federated Grids [34, 61, 63]. In the case of federated Grids, the aim of the meta-broker, which collects information of the different Grid meta-brokers, is to select the most appropriate Grid system to execute the different tasks.

The different local resources are composed of a set of heterogeneous machines, which are assumed to provide different computing capabilities, have multiple cores, and different power consumption depending on the hardware features. Both the execution time for running an individual task and the energy consumption to perform it vary from one machine to another. Consequently, those machines that are able to execute the tasks in the shortest time and/or with the lowest energy consumption will have high demand in the system.

Tasks are assumed to be independent, and they arrive to the meta-scheduler from the different users of the Grid, who collects and maps them into the resources of the system with a given frequency to optimize some criteria. The independent tasks model typically arises in Grid and volunteer-based computing infrastructures—such as Teragrid, WLCG, Berkeley's BOINC, Xgrid, etc. [11]–, where non-dependent applications using domain decomposition are very often submitted for execution. In addition, the independent tasks model also arises in the case of Bag-of-Task (BoT) applications. They are used in a variety of scenarios, including Single-Program Multiple-Data applications used for multimedia processing, parameter sweeps, data mining, parallel domain decomposition of numerical models for physical phenomena, simulations, and computational biology. Thus, the relevance of the scheduling problem faced in this work is justified due to its significance in realistic distributed HC and Grid environments.

In our model, tasks have variable computing demands, and they can be allocated into any machine in the system. Tasks are assumed to be the atomic unit of workload, so they cannot be divided into smaller chunks. Additionally, we consider a *non-preemptive* behavior, so tasks cannot be interrupted once they have been assigned to a machine.

We consider scheduling systems where all independent tasks arrive in the system in batches. When a batch arrives, the set of tasks composing it are scheduled and queued in the assigned resources. All tasks that arrive during the time interval between two batches will be processed in the next batch, and scheduled according to the resources availability. Different batch schedulers handle the tasks' information in a conservative way: at a given time, all decisions are taken assuming that the available information is exact. When an external event occurs, the scheduling process is restarted with the new state of the system, without rescheduling the already running tasks [22]. This allows the transformation of the online scheduling problem to be solved in an offline fashion [32]. A relation between this framework and the framework where tasks are released over time is studied for different scheduling strategies for general or restricted cases. There are several production systems in use whose primary focus is job batching and resource scheduling, like OAR [5], Portable Batch Systems (PBS) [3], Load Sharing Facility (LSF) [4], and Condor [1]. They are supported as a job scheduler mechanism by several

meta-schedulers, like Grid Resource Allocation Manager [2].

The meta-scheduler targets the minimization of the execution time and the energy used. The most usual strategy to accomplish the former goal is to minimize the *makespan*, defined as the time spent from the moment when the first task begins execution to the moment when the last task is completed [46]. Regarding the energy consumption, it is computed according to the novel model we present in Section 3. These two objectives are usually in conflict, meaning that for a given optimal solution, reducing energy could worsen the QoS provided by the system. This QoS agreement must be set together by the user and the provider. However, the provider will always be interested on finding schedules with minimized energy requirements for a given QoS level.

We tackle in this work the bi-objective heterogeneous computing scheduling problem, aimed at minimizing the makespan and energy consumption (ME-HCSP). The mathematical formulation for the problem considers the following elements:

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

The goal of the ME-HCSP is to find an assignment function $f : T^N \rightarrow P^M$ which simultaneously minimizes the *makespan* metric, defined as follows: any task is scheduled without preemption from time $ST(t_i)$ on machine m_j, with an execution time given by $ET(t_i, m_j)$. The task t_i completes at time $C_i = ST(t_i) + ET(t_i, m_j)$. The makespan is the maximum completion time $C_{\max} = \max(C_i)$ and the *total energy consumption*, defined in (1).

$$\sum_{\substack{t_i \in T: \\ f(t_i) = m_j}} EC(t_i, m_j) + \sum_{m_j \in P} EC_{\text{IDLE}}(m_j) \tag{1}$$

The total energy consumption accounts for both the energy required to execute the assigned tasks, and the energy that each machine consumes in idle state (see a descriptive example in Fig. 2). Therefore, we do estimations for the worst case since, in real systems, idle machines can be changed to an energy saving mode (or switched off) or they can be used to compute the next arriving set of tasks.

The energy required to execute a given task depends on both the execution time $ET(t_i, m_j)$ and the machine it is assigned to. Additionally, it also depends on the resource utilization: it might be higher when the server is underutilized (i.e., when the number of assigned tasks is less than the available cores) than in the case when all cores are busy with tasks. Therefore, the two objectives (i.e., makespan and energy consumption) are not directly related, and multi-objective optimization seems an appropriate approach to solve the problem.

In the previous formulation all tasks can be independently executed, disregarding the execution order. As previously mentioned, the independent task model is common in Grid and

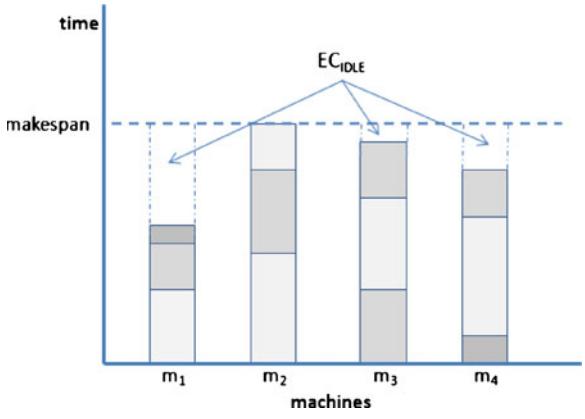


Fig. 2 Energy consumption includes the energy in idle state

volunteer-based computing infrastructures, as well as in BoT applications.

Next section introduces the novel model we propose in this paper for computing the energy consumption in a heterogeneous system of multicore computers.

3 Model for Energy Consumption in Multicore Computers

The main differences between the problem formulation in the previous section and other existing models are: i) the energy consumption is considered along the makespan as simultaneous objectives in the scheduling process, and ii) the HC system is composed of *multicore* computers, each of them with a specific power consumption, depending on the number of cores used.

This section introduces the new model proposed to compute the energy consumption for a tasks-to-processsors assignment in a multicore HC Grid system.

3.1 Data Used in the Model: Computing Resources

The proposed model characterizes the computing resources in the HC Grid system by four parameters: i) the computing power of a machine, i.e. the number of operations that its processor is able to compute; ii) the number of processing cores that the processor integrates (*cores*); iii) the energy consumption when the processor is in idle state (E_{IDLE}); and iv) the energy consumption when the processor is fully loaded (E_{MAX}).

In order to take into account a realistic value for the computing power of a given machine, we decided not to use the maximum theoretical GFLOPS reported for each vendor, since this is a peak value that usually does not reflect the real computing capacities of the machine. Instead, we adopt the average value of operations per second reported when solving the standardized energy efficiency benchmark SSJ, from Standard Performance Evaluation Corporation (SPEC) [30, 67]. SSJ is the first industry-standard SPEC benchmark designed to evaluate both the power and performance characteristics of multinode/multicore computers. The benchmark defines power measurement standards based on a specific methodology [66], in the same way that SPEC has done for performance in the past.

Power consumption charts and processor data collected from the SPEC website are used to evaluate the energy consumption for a given schedule. Table 1 reports the reviewed data for each processor: processor name, clock frequency (in GHz), number of cores, number of operations per second when solving the SSJ benchmark (*ops*), energy consumption in idle sate, E_{IDLE} (in Watts), and energy consumption when all the processor cores are used, E_{MAX} (in Watts). Table 1 also reports the *operations per Watt* (*ops/W*) metric, which gives a more realistic approach to characterize resources for energy-aware computation, since the cost of powering a given processor rapidly outweighs the cost of the processor itself.

3.2 Data Used in the Model: Tasks and ET Function

The proposed formulation for the energy-aware scheduling problem in multicore HC Grid systems is based on a given ET function. In this work, the two models most widely used in literature to represent HC scenarios are considered: the *related machines* model [7] and the *unrelated machines* model [45].

In addition, we use a performance estimation model based on those two well-known methods to define the time needed to perform each task in the HC Grid system:

- Related machines model: each task demands a fixed number of operations $TO(t_i)$, disregarding the type of machine used. In this work, related machine scenarios are represented by defining the processing time of task i in machine j by $ET(t_i, m_j) = TO(t_i)/[op(m_j)/cores(m_j)]$, based on the approach by Zomaya and Teh [74].
- Unrelated machines model: each task demands a fixed number of operations $TO(t_i)$, but the execution in a specific machine also considers an additional overhead $AO(t_i, m_j)$. The function AO is used to model the most generic case for an HC scenario, where

Table 1 Energy and computing power for the reviewed processors

	Processor	GHz	# Cores	Ops	E_{IDLE} (W)	E_{MAX} (W)	Ops/watt
1	AMD Opteron 2216 HE	2.40	2	47,927	82.0	138.0	101.5
2	AMD Opteron 2356	2.30	4	122,114	79.3	150.3	521.5
3	AMD Opteron 2376 HE	2.30	4	173,163	59.5	105.0	1,044.0
4	AMD Opteron 2377 EE	2.30	4	171,052	36.6	86.0	1,379.0
5	AMD Opteron 2380	2.50	4	154,045	69.0	134.5	731.0
6	AMD Opteron 2382	2.60	4	165,629	63.5	129.0	851.0
7	AMD Opteron 2384	2.70	4	169,068	68.5	131.6	834.5
8	AMD Opteron 2419 EE	1.80	6	203,477	37.3	89.0	1,614.0
9	AMD Opteron 2425 HE	2.10	6	231,108	40.2	109.0	1,532.0
10	AMD Opteron 2435	2.60	6	256,780	61.0	129.5	1,350.5
11	AMD Opteron 4164 EE	1.80	6	211,874	30.7	67.8	2,043.0
12	AMD Opteron 6168	1.90	12	416,178	48.0	131.0	2,210.0
13	AMD Opteron 6174	2.20	12	459,729	40.5	134.7	2,481.1
14	AMD Opteron 6176	2.30	12	469,103	42.8	140.7	2,452.3
15	AMD Opteron 6262 HE	1.60	16	396,768	35.2	106.5	2,618.0
16	AMD Opteron 6276	2.30	16	570,390	35.8	152.8	2,868.3
17	AMD Opteron 8376 HE	2.30	4	153,377	58.0	105.7	938.3
18	Intel Core i3-540	3.07	2	136,634	25.0	61.7	1,556.0
19	Intel Core i7 610E	2.53	2	126,976	20.8	47.4	1,856.0
20	Intel Pentium D 930	3.00	2	52,303	105.0	169.0	190.0
21	Intel X3350	2.67	4	173,021	68.9	119.0	913.0
22	Intel Xeon 3040	1.86	2	54,479	86.0	117.0	268.0
23	Intel Xeon 3075	2.66	2	98,472	93.7	135.0	431.0
24	Intel Xeon 5160	3.00	2	82,084	101.8	151.5	328.0
25	Intel Xeon 7020	2.66	2	21,521	130.0	208.3	61.1
26	Intel Xeon 7110M	2.60	2	37,185	143.8	183.0	114.0
27	Intel Xeon E3110	3.00	2	118,486	75.2	117.0	605.0
28	Intel Xeon E5345	2.33	4	119,239	114.0	170.8	413.0
29	Intel Xeon E7330	2.40	4	126,597	104.5	156.8	489.0
30	Intel Xeon L3360	2.83	4	183,767	48.7	95.0	1,253.0
31	Intel Xeon L5335	2.00	4	111,872	107.3	146.0	443.0
32	Intel Xeon 3070	2.67	2	78,928	78.8	120.0	405.0
33	Intel Xeon 5160	3.00	2	79,576	86.0	129.0	382.0
34	Intel Xeon E3-1220	3.10	4	329,862	22.4	92.8	3,026.0
35	Intel Xeon E3-1260L	2.40	4	314,438	17.8	55.7	4,327.5
36	Intel Xeon E3-1270	3.40	4	394,356	21.4	102.0	3,265.0
37	Intel Xeon E3-1280	3.50	4	412,077	30.9	106.2	3,214.0
38	Intel Xeon E5420	2.50	4	143,516	77.5	130.0	681.0
39	Intel Xeon E5440	2.83	4	150,979	76.9	131.8	709.0
40	Intel Xeon E5462	2.80	4	161,999	65.0	126.0	854.0
41	Intel Xeon E5472	3.00	4	137,497	92.7	160.3	551.3
42	Intel Xeon E5540	2.53	4	275,475	34.5	110.3	1,807.5
43	Intel Xeon E7-4870	2.40	10	556,330	105.8	209.3	1,816.0
44	Intel Xeon E7-8870	2.40	10	602,771	126.5	246.8	1,586.0
45	Intel Xeon L3360	2.83	4	199,707	55.2	101.9	1,255.6
46	Intel Xeon L5335	2.00	4	100,845	93.5	130.0	446.0
47	Intel Xeon L5408	2.13	4	121,036	104.2	136.0	505.0
48	Intel Xeon L5410	2.33	4	144,002	64.1	105.0	838.0
49	Intel Xeon L5420	2.50	4	138,462	63.3	105.1	835.1
50	Intel Xeon L5430	2.67	4	152,533	68.9	108.8	870.9
51	Intel Xeon L5520	2.26	4	228,603	31.8	89.5	1,791.5
52	Intel Xeon L5530	2.40	4	251,057	32.3	89.5	1,981.0
53	Intel Xeon L5630	2.13	4	226,022	30.4	69.0	2,118.0

Table 1 (continued)

	Processor	GHz	# Cores	Ops	E_{IDLE} (W)	E_{MAX} (W)	Ops/watt
54	Intel Xeon L5640	2.27	6	318,441	35.7	96.1	2,562.6
55	Intel Xeon L7345	1.86	4	89,881	67.8	96.8	546.0
56	Intel Xeon X3220	2.40	4	143,742	79.8	132.0	667.0
57	Intel Xeon X3360	2.83	4	188,173	59.6	118.5	1,343.8
58	Intel Xeon X3470	2.93	4	307,615	38.2	120.0	2,051.2
59	Intel Xeon X3480	3.07	4	325,650	38.3	121.5	1,912.9
60	Intel Xeon X5272	3.40	2	104,222	93.0	140.0	450.0
61	Intel Xeon X5570	2.93	4	280,253	41.2	122.8	1,873.8
62	Intel Xeon X5670	2.93	6	443,655	37.2	126.1	2,656.4
63	Intel Xeon X5675	3.07	6	459,263	59.9	150.4	2,426.2
64	Intel Xeon X7560	2.27	8	461,606	135.6	241.5	1,239.6

the values of $ET(i, j)$ are not related with each other, mainly due to contention, use of local/remote resources and data etc. In this work, unrelated machine scenarios are represented by defining the time to perform task i in machine j by $ET(t_i, m_j) = [TO(t_i) + AO(t_i, m_j)]/[op(m_j)/cores(m_j)]$, following the *expected time to compute* (ETC) estimation model [6].

The values of $ET(i, j)$ are supposed to be an input for the energy consumption model. Scheduling strategies in multicore computers have to take into account that using all the cores available in each machine significantly reduces the energy consumption, as an idle processor demands between 40 to 80 % of the energy consumption when the full processor is used.

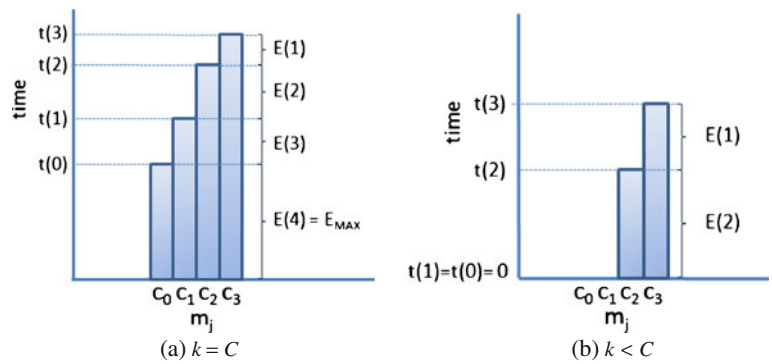
3.3 How to Compute the Energy Consumption

A machine-based representation is used for computing the energy consumption of schedules. Without losing generality, the cores within a given machine are ordered considering the sum of ET for the tasks assigned to it (i.e., the total computing time of core c_0 is lower than the total computing time of core c_1, and so on).

A linear model has been adopted for the energy consumption, according to the one by Minas and Ellison [53] (2). The energy consumption linearly varies from E_{IDLE} when the processor is idle, to E_{MAX} when the processor is fully used; k is the number of cores used, C is the number of cores in the machine, and $E(k)$ is the energy consumed when using k cores.

The proposed machine representation allows computing the energy consumption for every machine in the HC Grid system. By sorting the cores regarding their processing time, the energy consumption for a given machine is computed by (3), where $t(c_i)$ denotes the sum of ET for the tasks assigned to core i.

$$E(k) = E_{IDLE} + (E_{MAX} - E_{IDLE}) \times \frac{k}{C} \quad (2)$$



(a) $k = C$ (b) $k < C$

Fig. 3 Computing the energy consumption for machine m_j

$$
\begin{aligned}
& t(0) \times E_{\mathrm{MAX}} \\
& \quad + \sum_{h=1}^{C-1} E(C-h) * [t(h) - t(h-1)] \quad \text{if } k = C \\
& t(C-k) \times E(k) + \sum_{h=1}^{k-1} E(k-h) \\
& \quad \times [t(C-k+h) - t(C-k+h-1)] \text{if } k < C \quad (3)
\end{aligned}
$$

Figure 3 exemplifies the energy consumption calculation for a given schedule in a machine m_j, considering the cases of a fully loaded machine ($k = C$, in (a)) and a partially loaded machine ($k < C$, in (b)), respectively.

4 Related Work

This section reviews the main related work about existing list scheduling heuristics and energy optimization in Grid scheduling.

4.1 List Scheduling Heuristics

The class of *list scheduling techniques* comprises a large set of deterministic static scheduling methods that work by assigning priorities to tasks based on a particular ad-hoc heuristic. After that, the list of tasks is sorted by decreasing priority and each task is assigned to a processor, regarding the task priority and the processor availability. Algorithm 1 presents the general schema of a list scheduling method.

Since the pioneering work by Ibarra and Kim [33], where the first algorithms following the schema in Algorithm 1 were introduced, many list scheduling techniques have been proposed in order to provide easy methods for tasks-to-processors scheduling.

The simplest category of list scheduling heuristics applied to minimize the makespan metric uses a single criterion to perform the tasks-to-machine assignment. Many popular heuristics fit into this category, including SJFR, LJFR, OLB, MCT, and others [13, 24].

Trying to overcome the inefficacy of these simple heuristics, other list scheduling methods have been proposed taking into account more complex and holistic criteria to perform the task mapping, and then reduce the makespan values [13]. Some of the most popular heuristics in this class include:

- *MinMin*: greedily picks the task that can be completed the soonest. The method 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 best machine. The mapped task is removed from U, and the process is repeated until all tasks are mapped. MinMin does not consider a single task at a time but all the unmapped tasks sorted by MCT, and the availability status of the machines is accordingly updated after every assignment. This procedure generally leads to more balanced schedules (i.e., better makespan values) than other heuristics, since more tasks are expected to be assigned to the machines that can complete them the earliest;
- *MaxMin*: is similar to MinMin, but it assigns the task with the overall *maximum* MCT to the best machine. Therefore, larger tasks are allocated first in the most suitable machines and shorter tasks are mapped afterwards, trying to balance the load of all machines. In some scenarios, MaxMin can provide better schedules than MinMin;
- *Sufferage*: identifies in each iteration the task that will *suffer* the most if it is not assigned to a certain host. The *sufferage value* is computed as the difference between the best MCT of the task and its second-best MCT. Sufferage gives precedence to those tasks with high sufferage value, assigning them to the machines that can complete them at the earliest time. This approach could lead to find better schedules than MinMin and MaxMin.

The MinMin, MaxMin, and Sufferage heuristics follow a generic schema which applies two *phases* to perform the task-to-resource assignment: in the first phase, N pairs (task, machine) are selected considering a specific criterion, and then in the second phase one of the N pairs is selected regarding an overall comparison. Traditional list scheduling heuristics were conceived for optimizing only one objective function, i.e. the makespan of the schedule, thus the criteria applied in the two optimization phases are related to the makespan

Algorithm 1 Schema of a list scheduling algorithm

```
1: while tasks left to assign do
2:     determine the most suitable task according to the chosen criterion
3:     for each task to assign, each machine do
4:         evaluate criterion (task, machine)
5:     end for
6:     assign the selected task to the selected machine
7: end while
8: return task assignment
```

metric. In this article, we extend this approach by considering the two objective functions considered in the bi-objective HCSP version tackled, i.e. makespan and energy consumption (see Section 5).

4.2 Energy Models in Computational Grids

We review in this section some of the most relevant energy models for computational Grids in the literature. Probably, the simplest approach found in literature is to switch off unused resources. Orgerie et al. [55] presented a centralized On/Off algorithm for the resources managed by the scheduler, which is organized hierarchically at node, cluster, and sites levels. The energy model considers the power consumption of a given resource in an idle state, when running at full capacity, when it is Off, and also the required energy consumption to switch between On and Off modes, or vice versa. The model was later extended in [56] with predictions on nodes usage and reservations of unused nodes in order to avoid frequent On/Off cycles. The framework by Da Costa et al. [16] is based on three components: an On/Off model based on an energy-aware resource infrastructure which collects energy logs from distributed energy sensors, a resource manager system which provides a workload prediction for automatic node shut down at a cluster level, and a trust delegation component to assume network presence of sleeping nodes.

The workload consolidation strategy by Hermenier et al. [31] uses a dynamic placement system for virtual domains to reduce power consumption on a Grid virtualized with Xen on each node. The placement policy concentrates the workload on a minimum set of nodes to shutdown unused resources with a negligible impact on performance. The energy-aware policy used is based on the switching On/Off modes and the energy model only accounts for the energy spend when a node is on an active state.

Coutinho and de Carvalho [15] designed three strategies for meta-scheduler resource allocation to reduce energy consumption on a Grid of several distributed HC sites. The authors consider that Grid machines are turned on all the time, and their load varies on time, determining their energy consumption. The energy model only accounts for the consumption of resources according to their load.

Kumar and Buyya [28] faced the problem at the meta-scheduling level by distributing compute-intensive parallel applications. The Grid resource sites consist of clusters at different geographical locations. The energy model accounts for the consumption of resources, considering dynamic and static power, and the cooling system. The static power includes the base power consumption of the CPU and all other components. The CPUs support DVS facility and thus their frequency can be varied in a predefined range. Hence, at a local resource site, the scheduler uses DVS policy to further reduce energy consumption.

Fei et al. [71] tackled the energy optimization problem in Grids by using load-balancing and switching idle resources into sleeping mode. The load-balancing strategy considers the job migration cost, resource heterogeneity, and network dynamics for a Grid composed of a set of sites with different node capacity. The energy model accounts for the consumption at active mode, at startup period, at the sleep mode, and at the idle mode.

In addition to the previous works, a there are a few studies addressing the energy optimization at the level of multi-cores in computational Grids. Lammie et al. [42] explored energy and performance tradeoffs in the scheduling of Grid workloads. The authors combine frequency scaling for reducing speed of the CPU at single nodes (i.e. scaling all the cores to the same CPU frequency,

either the highest or the lowest supported frequencies), automated node scaling to power off idle resources, and intelligent job assignment to evaluate and optimize assignments. The target architecture is composed of identical multi-core nodes, and the energy consumed by each node under the same load is also identical. All jobs are strictly CPU intensive and use 100 % of the CPU cycles assigned to them.

Barrondo et al. [8] considered a hierarchical model for the power consumption on a P2P Grid system at core, node, cluster, and Grid levels. The power model assumes that power consumption of all system components is essentially constant, regardless of the machine activity. Off cores consume no power, while On cores consume power when they are either in idle or operational modes. At a node level, a startup duration with power consumption is used when turning on and off each component. The total energy consumed by the Grid is the sum of the starting up energy and the energy it consumes while its operational state.

Delver et al. [18] focused on reducing energy consumption of Grid server sites. The energy saving strategy is based on the dynamic power deactivation of underloaded servers combined with proper task scheduling. The power consumption of a server is captured in a linear model. The authors consider multi-core processors. A certain amount of power is consumed even if the server is in idle state. For each of the cores, the additional power increases linearly with the load for a fully loaded core. The model also accounts for additional amount of power of network communications, a startup power and power for operational mode.

Most of the previous works consider dynamic power deactivation, DVS, or other power technique to optimize energy consumption at the Grid level. In this paper we do not use these techniques. Instead, we propose to optimize energy at the level of the scheduler, making use of the heterogeneity of resources.

4.3 Energy-Aware Scheduling Methods

Task scheduling with energy efficiency considerations has been addressed in the related literature by using two main approaches: independent and simultaneous.

In the independent approach, a *best-effort* scheduling method is combined with a slack reclamation technique. Energy and performance constraints are assumed independent, and new schedulers or existing schedulers are adapted to become energy-efficient [9, 10, 62].

In the simultaneous approach, the scheduling is computed with energy saving considerations to satisfy both QoS and energy constraints simultaneously. A multi-constrained bi-objective model is used for the problem, and the goal is to find Pareto optimal schedules such that no other schedule can execute all the tasks faster and using less energy. Khan and Ahmad [35] studied the problem of allocating independent jobs on a Grid to simultaneously minimize schedule length and energy. The authors proposed a method based on the concept of Nash Bargaining Solution from cooperative game theory, assuming DVS-enabled machines.

Kim et al. [36] addressed the problem of scheduling tasks with different priorities and deadline constrained in an ad hoc Grid environment where resources have limited battery capacity with DVS for power management. The authors introduced several online and batch mode dynamic heuristics, based on the well-known MinMin method [51]. The results showed that batch mode dynamic performed the best. However, they also required significantly more time.

Li et al. [48] presented an online dynamic power management strategy with multiple power-saving states. Then, they proposed an energy-aware scheduling algorithm to reduce energy consumption. The proposed algorithm is based on MinMin. Pinel et al. [58] proposed a two-phase heuristic for the scheduling of independent tasks on Grids with energy considerations. First, MinMin is applied to find good makespan, then a local search algorithm is applied in the second phase.

The semi-static cooperative, power-aware game theoretic scheduler by Subrata et al. [68] takes into account the ownerships resource providers and incentive for cooperation and simultaneously optimize the utility of the providers on a federated Grid. The energy consumption is based on the speed scaling model; the lower

the speed, the higher the saving in energy cost. However, the speed of resources depends on the target expected response time. The goal is to optimize not only the total power saving, but also maximize savings between all the providers.

A recent work by Lindberg et al. [49] tackles the Grid scheduling problem, aiming at minimizing makespan and energy subject to deadline constraint and tasks' memory requirements. A set of eight heuristics were introduced, including six greedy algorithms based on the list scheduling approach and two GAs.

5 Heuristics for the Bi-Objective ME-HCSP Optimization

This section introduces the new scheduling heuristics for the optimization of the makespan and energy consumption in multicore HC Grid systems.

Such heuristics will have important benefits versus other methods dealing with aggregated functions and Pareto based algorithms. On the one hand, the techniques using aggregated functions require the definition of the weights to look for the desired tradeoff solution, which is not an easy task. Additionally, such approaches are not able to explore concave regions of the Pareto front, whatever the weights defined. On the other hand, Pareto based approaches require a decision maker to choose one among all the provided non-dominated solutions, making it more complex to design a fully automatized scheduler. Moreover, such algorithms have strong computational requirements to generate diverse and accurate solutions, and they require their population to be seeded in order to find accurate results in reasonable time [19, 72].

Therefore, the proposed heuristics arise as an appropriate solution for the problem at hands because of the accurate solutions they provide with low computational and time requirements. The reason is that they directly look for solutions in the desired trade-off region, unlike Pareto based approaches, that require high computational efforts to find a diverse set of solutions, from which later on we select the desired trade-off one. Additionally, these heuristics can be used in parallel to seed the population of a more complex multi-objective evolutionary algorithm in the case that highly accurate solutions are required for the problem.

5.1 General Description of the New Scheduling Heuristics

All the proposed heuristics work in two phases, following the generic schema described in Section 4. Two criteria have been considered, taking into account the makespan and the energy consumption optimization.

The generic method for scheduling in the new proposed heuristics is summarized as follows. In phase 1, the method selects for each task the machine that better accomplishes the first selected criterion. Then, in phase 2, among the pairs (task, machine) computed in phase 1, the method selects the pair that better accomplishes the second criterion.

In addition, specific variants of the new heuristics have been implemented by only taking into account the *k-percent-best* machines. The *kpb*-versions of the scheduling heuristics find for each task the k percent of the machines with the minimum execution time for that task, ignoring the machine availability, and also ignoring the MCT for the task. Then, the first phase in the list scheduling algorithm is performed only by considering this subset of machines for each task, while no modification is done when applying the second phase.

A total number of twenty heuristics were designed considering the makespan and energy consumption criteria and their *kpb* variations. The new heuristics have been named using the following convention:

```
[kpb]<2nd optim. criterion><1st optim.
criterion>
```

In the previous name convention, those criteria which optimize the makespan metric are written in lowercase (e.g.: Min, Max, Suff), and those criteria that optimize the energy consumption are written in uppercase (e.g.: MIN). Those strategies that use a k-percent-best list have the suffix *kpb* -written in lower case-, indicating that a list of machines sorted by minimum execution time is used in the proposed heuristic. The implemented heuristics are described in the next subsection.

5.2 New Scheduling Heuristics

The new scheduling heuristics for the bi-objective optimization of makespan and energy consumption in HC and Grid computing systems can be grouped into five categories:

1. **Traditional heuristics**, which only optimize the makespan metric, but in this case they also calculate the energy consumption for the resulting schedule: *MinMin*, *MaxMin*, and *Sufferage*, already described in Section 4.1.
2. **Greedy energy**: *MINMIN*, which only focuses on optimizing the energy consumption in the two phases of the list scheduling schema, and after that it computes the makespan of the resulting schedule.
3. **Combined heuristics**, which combine the makespan and energy consumption optimization criteria:
   - *MINMin*: selects the pairs (task, machine) that minimizes the MCT (in phase 1), and then selects the pair that minimizes the energy consumption (in phase 2).
   - *MINSuff*: selects the pairs (task, machine) that minimizes the sufferage value for each task (in phase 1), and then selects the pair that minimizes the energy consumption (in phase 2).
   - *MinMIN*: selects the pairs (task, machine) that minimizes the energy consumption for each task (in phase 1), and then selects the pair that minimizes the MCT (in phase 2).
   - *MaxMIN*: selects the pairs (task, machine) that minimizes the energy consumption for each task (in phase 1), and then selects the pair that maximizes the MCT (in phase 2).
   - *SuffMIN*: selects the pairs (task, machine) that minimizes the energy consumption for each task (in phase 1), and then selects the pair that minimizes the sufferage value (in phase 2).
4. **Alternated heuristics**, which alternate the search proposed by traditional and combined heuristics (one task is assigned following the first heuristic, the next one following the other, and so on). The alternated heuristics were designed considering the traditional and combined heuristics that achieved the most promising results in preliminary experiments:
   - *MinMIN-MINMin*: alternates the MinMIN and MINMin heuristics.
   - *Sufferage-MINMin*: alternates the Sufferage and MINMin heuristics.
   - *MINMIN-MinMin*: alternates the MINMIN and MinMin heuristics.
   - *MinMIN-MinMin*: alternates the MinMIN and MinMin heuristics.
   - *MaxMIN-MINMin*: alternates the MaxMIN and MINMin heuristics.
5. ***k*-percent-best**, the variants of proposed heuristics that use a *kpb*-list (after performing an experimental evaluation, the size of the *kpb*-list was fixed to 75 % of the machines in each scenario):
   - *kpb-MinMin*: works like MinMin, but using a list of the *k*-percent-best machines for each task.
   - *kpb-Sufferage*: works like Sufferage, but using a list of the *k*-percent-best machines for each task.
   - *kpb-MaxMin*: works like MaxMin, but using a list of the *k*-percent-best machines for each task.
   - *kpb-MinMIN*: works like MinMIN, but using a list of the *k*-percent-best machines for each task.
   - *kpb-SuffMIN*: works like SuffMIN, but using a list of the *k*-percent-best machines for each task.
   - *kpb-MINSuff*: works like MINSuff, but using a list of the *k*-percent-best machines for each task.

Figure 4 provides sample pseudocodes for four of the proposed heuristics, one in each new category: MINMIN for greedy energy, MINSuff for combined, MinMIN-MINMin for alternated, and *kpb*-MinMIN for *k* percent best. The other heuristics in each category work in a similar way than the corresponding example, but using different criteria for the search in the inner iterations.

Fig. 4 Sample pseudocodes for four of the proposed heuristics

MINMIN (greedy energy)

```
1: U = T{set of unsigned tasks}
2: while U ≠ ∅ do
3:    for each task t_k ∈ U do
4:       for each machine m_h ∈ P do
5:          evaluate EC (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 EC
10:   Assign t_i to m_j; remove t_i from U
11: end while
12: return task assignment
```

MINSuff (combined)

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

MinMIN-MINMin (alternated)

```
1: U = T{set of unsigned tasks}
2: while U ≠ ∅ do
3:    if #(U) mod 2 then
4:       Assign best task using MinMIN
5:    else
6:       Assign best task using MINMin
7:    end if
8:    Remove best task from U
9: end while
10: return task assignment
```

***kpb*-MinMIN** (k-percent-best)

```
1: U = T{set of unsigned tasks}
2: while U ≠ ∅ do
3:    for each task t_k ∈ U do
4:       for machine m_h ∈ kpbList(t_k) do
5:          evaluate EC (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 MCT
10:   Assign t_i to m_j; remove t_i from U
11: end while
12: return task assignment
```

6 Experimental Analysis

This section presents the experimental evaluation of the twenty heuristics proposed to optimize makespan and energy consumption in multicore HC Grid systems.

6.1 ME-HCSP scenarios

There has been little effort to define standardized problem benchmarks or test suites for the HCSP. Braun et al. [13] presented a test suite of twelve instances with 512 tasks and 16 machines using the ETC model [6]. These suite have become a *de facto* standard benchmark for evaluating algorithms to solve the HCSP.

In this work, a set of ME-HCSP instances has been built to model both unrelated and related machines scenarios. The problem instances are defined by a *workload file* with the task information, and a *scenario file*, with the relevant data for each machine in the HC system.

Each scenario file describes a specific HC infrastructure, by defining the relevant features for each machine in the system: the number of cores, the number of operations, and the values of E_{IDLE} and E_{MAX} from the SSJ SPEC benchmark results.

The scenarios are classified in three categories of *machine heterogeneity*, regarding the standard deviation in the SSJ operations per core: low (<36 %), medium (36–44 %), and high heterogeneity (>44 %).

In addition, two classes of workload files are used:

- Related machines model: the workload files contain the fixed number of operations $TO(t_i)$ for each task. The values of $TO(t_i)$ are generated by randomly sampling using a uniform distribution in a given range, following the model by Zomaya and Teh [74]. The heterogeneity classification does not apply for these instances, since the workload files do not include overhead for each machine (one single heterogeneity class is used for the related machine model).
- Unrelated machines model: the workload files have the number of operations for each task, defined by $TO(t_i) + AO(t_i)$. The values of $TO(t_i)$ are generated by randomly sampling using a uniform distribution in a given range, and the values for $AO(t_i)$ are a random percentage of $TO(t_i)$. This methodology for workload generation is similar to the one used

in the range method of the ETC model [6] to create the HCSP benchmark instances by Braun et al. [13]. The range for $TO(t_i)$ is (0, 1e+10), based on the lengths reported for tasks executed in our high performance computing system, the cluster FING [54], one of the National Grid Initiatives included in the GISELA Grid infrastructure [29]. Three *task heterogeneity* ranges are defined for $AO(t_i)$: low heterogeneity (5–20 %), medium heterogeneity (5–35 %), and high heterogeneity (5–50 %).

In this work, we studied a large number of ME-HCSP instances with dimension (tasks × machines) 512 × 16, 1,024 × 32, and 2,048 × 64. For each dimension, we generated 20 scenario files and 40 workload files, i.e. 800 problem instances combining scenario/workload for each dimension, and 2,400 problem instances in total. By studying this very large number of workloads and scenarios, we are able to perform an exhaustive experimental evaluation of the proposed energy-aware scheduling heuristics over very different instances modeling very diverse situations, in order to draw representative conclusions for nowadays Grid infrastructures.

Table 2 reports the average number of cores in the scenarios for each problem dimension, and the average machine heterogeneity (quotient of standard deviation of ssj_ops/core and average value of ssj_ops/core). Regarding both the number of tasks and effective computing resources (cores), the instances solved in this work have a reasonable large size when compared with those usually used to evaluate scheduling heuristics for HC and Grid systems.

The ME-HCSP scenarios used in this article were generated using a python script and the Mersenne Twister pseudorandom number generator. The generator script and the scenarios are publicly available at http://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP.

Table 2 Details of the scenarios generated for each problem dimension

Dimension	Avg. # cores	Avg. heterogeneity (%)
512 × 16	73.2	40.1
1,024 × 32	138.8	41.7
2,048 × 64	314.8	49.1

6.2 Development and Execution Platform

The scheduling heuristics were implemented in C, using the standard `stdlib` library and GNU gcc. The experimental analysis was performed on a Xeon E5430 processor at 2.66 GHz, 8GB RAM, and the CentOS Linux 5.2 operating system (platform website: http://www.fing.edu.uy/cluster).

6.3 Experimental Results

This subsection summarizes and analyzes the main results of the experimental evaluation of the proposed heuristics. The use of the state-of-the-art MinMin, MaxMin, and Sufferage heuristics allows the validation of the novel proposed algorithms in terms of makespan value. The MinMin schema targeting the minimization of the energy used instead of makespan (MINMIN) provides an accurate schedule in terms of energy use.

In order to analyze the differences in the makespan and energy results across the multiple instances solved for each ME-HCSP dimension, the *average relative ranking* for each heuristic was computed using the non-parametric Friedman statistical test for each objective. For example, a makespan rank value of 15 means that a given method occupied, on average, the 15th position when sorting the twenty heuristics regarding the makespan values (the average is computed over the 200 instances in each instance class). A similar procedure was used for the energy rank values.

We report the χ^2 and the p-values for the Friedman statistical test in each case. We test if the p-values are less than 10^{-3}, in order to state that there exists enough evidence to conclude that the results distributions differ in their true medium makespan/energy values for the studied heuristics, with 99 % confidence.

We also evaluated the relative difference (*GAP* metric) between the makespan/energy computed using each heuristic and the best result for each problem instance.

From the point of view of multiobjective optimization, we analyze the *dominance*, reporting

for each heuristic the average number of times the other heuristics are dominated (i.e. they compute both worst makespan and energy results) by the considered one.

6.3.1 Heuristics Comparison

The full results for the proposed heuristics are presented in Table 3. The tables report the average GAPs for the 200 instances solved for each scenario model and workload heterogeneity class (the best values in each case are marked with bold font). Figures 5 and 6 graphically compare the GAP values for makespan and energy averaged over all workloads and scenarios for each problem dimension tackled.

The results in Table 3 indicate that for the 512×16 ME-HCSP instances, the traditional MaxMin is the best heuristic for minimizing the makespan among the twenty schedulers studied, and Sufferage is consistently the second best option. This fact confirms classic results on scheduling stating that the best strategy to minimize the overall computing time is to execute the longest tasks on the fastest machines [12]. However, when accounting for the energy consumption, the combined heuristics SuffMIN, MaxMIN, MinMIN, the greedy MINMIN, and some *kpb* variants are able to compute more efficient schedules than MaxMin and other traditional heuristics.

The comparison also shows that MaxMin achieves its best energy results for the related machines model, but it tends to compute worse energy values for high heterogeneity scenarios. On the other hand, SuffMIN, MaxMIN and MINMIN perform better when optimizing energy on high heterogeneity instances. Thus, those three newly proposed heuristics are useful options to significantly reduce the energy consumption when executing in high heterogeneity ME-HCSP instances.

The results for the large ME-HCSP instances confirm the main conclusions drawn from the analysis of the 512×16 instances: MaxMin is the best method for minimizing the makespan among the twenty heuristics studied, and Sufferage and MaxMin-MINMin are steadily the second best options. Regarding the energy consumption, the combined heuristics MaxMIN and SuffMIN consistently obtained the best energy values overall, and MinMIN computed accurate energy values for high heterogeneity instances.

From the point of view of multiobjective optimization, the twenty heuristics provide schedules with different trade-offs between makespan and energy consumption. Table 4 reports for each heuristic, problem dimension, machine model, and heterogeneity class, the average number of times the other heuristics are dominated by the considered one.

The results in Table 4 indicate that Sufferage, MaxMin, and MaxMin-MINMin have the largest number of dominating solutions overall, mainly due to the accurate makespan values achieved. Regarding the energy consumption objective, Sufferage-MINMin, MaxMin-MINMin, and some *kpb* variants also report acceptable dominance values, especially for the alternated heuristics on the related machine model. Other methods, such as MINMin, MINSuff, and MINMIN-MinMin also show acceptable dominance values, especially for the largest instances tackled.

Figure 7 shows a Pareto comparison of the heuristics for four sample instances, which offer representative results for diverse workloads (WL) and scenarios (SC). Non-dominated schedules are marked in red. The analysis of Fig. 7 shows that the traditional MaxMin and Sufferage heuristics, as well as the new combined heuristics MaxMIN, SuffMIN, MaxMIN, and MinMIN, and also *kpb*.Sufferage, generally obtain the best compromise solutions for the four sample instances. A similar behavior was detected for all the other problem instances solved.

6.3.2 Scalability Analysis for Large Instances

The previous experimental analysis considered ME-HCSP instances with a fixed ratio between tasks and machines (32), according to the model used in the related literature [6, 13].

However, that setting can be unrealistic in order to model nowadays HC and Grid infrastructures, where eventually a larger number of tasks is submitted to execute on a given machine scenario. The scalability analysis reported in this subsection studies the efficacy of the best six heuristics identified using the Friedman rank tests reported

Table 3 Comparison of the proposed heuristics

	Related machines				Unrelated machines											
	Makespan		Energy		Low heterogeneity				Medium heterogeneity				High heterogeneity			
					Makespan		Energy		Makespan		Energy		Makespan		Energy	
	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)
512 × 16 ME-HCSP instances																
1-SuffMIN	15.1	25.3	1.9	2.7	19.0	37.9	1.6	1.6	19.1	41.9	**1.7**	**1.2**	19.1	43.2	**2.1**	**1.6**
2-Sufferage-MINMin	4.2	7.9	9.2	13.0	6.5	7.5	11.7	12.3	7.3	5.7	12.1	11.1	7.4	4.7	12.4	10.5
3-Sufferage	2.4	4.8	6.8	11.6	2.8	3.3	6.9	10.4	2.8	2.0	7.7	9.5	3.2	1.7	8.7	9.2
4-MINSuff	7.2	10.8	12.4	14.4	6.8	7.6	12.4	12.5	7.1	5.3	12.0	11.0	7.0	4.3	12.1	10.3
5-MinMIN-MINMin	14.9	21.5	18.3	21.0	15.3	18.4	18.9	18.9	15.7	16.7	19.1	17.6	15.8	16.1	19.1	17.1
6-MinMIN-MinMin	14.9	21.3	18.4	20.8	15.1	18.2	18.8	18.7	15.1	15.2	18.6	16.7	15.2	14.6	18.5	16.1
7-MINMIN-MinMin	7.4	11.6	12.9	14.8	6.9	7.9	12.1	12.4	6.3	5.3	10.0	10.6	6.1	4.1	9.8	9.8
8-MinMin	7.6	11.5	13.0	14.7	7.2	8.1	12.5	12.5	6.2	5.2	10.0	10.6	6.0	4.1	9.5	9.7
9-MinMIN	14.9	24.1	6.0	11.0	12.7	17.2	4.4	8.6	12.5	12.5	4.3	6.9	12.4	10.4	4.3	6.2
10-MINMin	6.2	10.8	11.4	14.4	5.8	7.6	11.4	12.5	6.1	5.3	11.0	11.0	6.0	4.3	11.1	10.3
11-MINMIN	15.7	25.3	6.3	11.2	16.2	22.7	4.9	9.0	16.3	20.7	5.2	7.4	16.2	19.6	4.9	6.5
12-MaxMin-MINMin	2.9	5.6	7.8	12.2	2.8	4.2	12.9	12.5	3.8	3.4	16.4	12.9	5.2	3.3	16.8	13.3
13-MaxMin	**1.7**	**0.8**	5.2	10.5	**1.1**	**0.2**	8.2	11.3	**1.2**	**0.4**	15.3	12.4	**1.7**	**0.7**	16.6	13.1
14-MaxMIN	12.2	20.7	**1.3**	**0.6**	15.6	22.1	**1.5**	**1.2**	17.0	23.6	1.8	1.9	17.4	25.2	**2.1**	2.6
15-*kpb*.SuffMIN	18.8	32.4	10.6	13.5	19.9	45.2	13.2	13.8	19.9	47.7	10.2	10.8	19.9	48.3	7.9	8.0
16-*kpb*.Sufferage	9.7	14.6	11.6	14.3	8.0	9.0	7.8	10.9	7.3	5.8	8.0	9.4	6.9	4.5	8.8	9.1
17-*kpb*.MINSuff	13.6	19.8	16.2	17.1	12.5	13.4	14.5	13.5	11.6	8.9	12.3	11.2	10.7	6.7	11.8	10.3
18-*kpb*.MinMIN	18.9	31.0	15.6	17.4	15.7	21.4	9.7	11.9	14.4	14.8	6.7	8.6	13.9	12.1	6.0	7.4
19-*kpb*.MinMin	13.8	20.0	16.3	17.2	12.2	13.2	13.9	13.2	11.2	8.7	11.4	10.9	10.1	6.4	10.3	9.8
20-*kpb*.MaxMin	7.8	12.1	9.1	13.3	7.9	9.0	12.7	12.7	9.0	7.3	16.3	13.2	9.9	6.6	17.3	13.8
χ^2	2757.5		3231.5		3420.8		678.7		3356.7		768.6		3211.4		1506.3	
p-value	$<10^{-3}$		$<10^{-3}$		$<10^{-3}$		$< 10^{-3}$		$<10^{-3}$		$<10^{-3}$		$< 10^{-3}$		$<10^{-3}$	

Table 3 (continued)

	Related machines				Unrelated machines											
	Makespan		Energy		Low heterogeneity				Medium heterogeneity				High heterogeneity			
					Makespan		Energy		Makespan		Energy		Makespan		Energy	
	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)
1,024 × 32 ME-HCSP instances																
1-SuffMIN	17.4	34.7	1.7	2.4	19.0	45.1	1.9	1.9	19.1	48.7	3.0	1.8	19.1	50.0	3.8	2.1
2-Sufferage-MINMin	4.8	9.1	9.6	13.8	7.2	8.1	12.3	10.7	8.0	6.6	12.8	9.4	7.6	5.4	11.9	8.9
3-Sufferage	2.6	5.4	6.5	12.2	3.0	4.3	7.3	9.1	2.7	2.6	7.5	7.8	3.3	2.3	8.3	7.7
4-MINSuff	6.8	10.4	11.7	14.5	6.0	7.1	11.4	10.3	6.8	5.7	11.8	9.1	7.0	4.8	11.6	8.7
5-MinMIN-MINMin	14.5	24.1	18.7	22.7	15.0	21.3	19.1	18.8	15.2	19.5	19.2	17.2	15.4	18.6	19.2	16.7
6-MinMIN-MinMin	14.8	24.2	19.0	22.8	14.9	20.6	19.0	18.3	15.1	19.1	19.2	16.9	15.3	18.0	19.1	16.3
7-MINMIN-MinMin	7.0	11.1	11.7	14.7	6.5	7.8	10.5	10.3	6.2	5.7	9.5	8.6	5.6	4.5	8.8	8.0
8-MinMin	7.2	11.0	11.7	14.7	6.8	7.8	10.9	10.4	6.4	5.8	9.7	8.7	6.4	4.8	9.3	8.1
9-MinMIN	14.9	30.8	8.4	13.5	13.5	23.4	7.1	8.1	13.6	16.9	3.6	5.2	13.2	14.0	3.2	4.3
10-MINMin	5.8	10.4	10.7	14.5	5.0	7.1	10.4	10.3	5.8	5.7	10.8	9.1	6.0	4.8	10.6	8.7
11-MINMIN	16.0	32.4	9.2	14.1	16.4	30.0	8.9	9.8	16.4	28.3	9.1	7.9	16.4	27.7	9.2	7.5
12-MaxMin-MINMin	2.5	5.3	6.4	12.2	2.5	3.7	10.7	10.3	2.9	2.9	15.4	10.3	3.5	2.8	16.0	10.7
13-MaxMin	**1.0**	**0.0**	3.6	10.0	**1.1**	**0.1**	7.1	9.2	**1.1**	**0.1**	13.4	9.8	**1.2**	**0.2**	15.2	10.3
14-MaxMIN	14.2	27.8	**1.4**	**1.3**	17.4	33.7	**1.4**	**0.8**	17.7	35.3	**1.6**	**0.9**	17.8	36.0	**2.0**	**1.6**
15-*kpb*.SuffMIN	19.3	41.1	14.5	17.9	20.0	51.9	15.9	15.7	19.9	53.8	13.6	11.9	19.9	54.7	12.6	10.7
16-*kpb*.Sufferage	10.1	15.9	10.0	14.6	8.0	9.0	6.7	8.7	6.5	5.6	6.1	7.0	6.1	4.4	6.7	6.8
17-*kpb*.MINSuff	12.3	20.8	15.3	17.5	11.6	13.1	12.5	11.2	11.6	9.3	11.7	9.2	11.3	7.8	11.6	8.7
18-*kpb*.MinMIN	12.3	38.6	15.3	21.4	11.6	27.2	12.5	11.7	11.6	19.1	11.7	7.1	11.3	15.9	11.6	5.8
19-*kpb*.MinMin	12.5	20.9	15.6	17.5	11.5	12.8	12.5	10.9	10.9	8.5	9.6	8.4	10.3	7.0	8.9	7.8
20-*kpb*.MaxMin	7.5	12.9	7.1	12.8	9.3	10.2	12.6	11.1	9.2	7.6	15.3	10.7	10.2	7.2	16.5	11.2
χ^2	3530.1		2946.5		3582.3		2777.3		3562.2		2589.2		3499.4		2674.8	
p-value	$<10^{-3}$		$<10^{-3}$		$<10^{-3}$		$<10^{-3}$		$<10^{-3}$		$<10^{-3}$		$<10^{-3}$		$<10^{-3}$	

Table 3 (continued)

	Related machines				Unrelated machines											
	Makespan		Energy		Low heterogeneity				Medium heterogeneity				High heterogeneity			
					Makespan		Energy		Makespan		Energy		Makespan		Energy	
	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)	Rank	GAP (%)
2,048 × 64 ME-HCSP instances																
1-SuffMIN	17.9	46.3	**1.5**	**1.9**	18.9	53.5	3.2	4.6	19.0	57.0	4.5	3.3	19.1	59.4	5.8	3.3
2-Sufferage-MINMin	5.1	10.8	8.1	14.0	7.3	11.1	11.2	10.2	7.3	9.2	11.2	7.4	7.1	8.3	11.0	6.4
3-Sufferage	2.5	5.5	4.8	11.6	3.3	6.1	5.7	7.9	3.4	5.3	6.2	5.7	3.2	4.6	5.7	4.8
4-MINSuff	5.9	11.5	9.0	14.4	5.7	9.5	9.3	9.5	6.2	8.3	10.8	7.2	6.8	7.8	11.5	6.3
5-MinMIN-MINMin	13.6	13.1	17.9	15.0	13.9	10.6	18.7	9.6	15.0	8.8	18.9	6.8	15.6	7.8	19.3	5.6
6-MinMIN-MinMin	14.0	28.4	18.3	24.7	14.0	26.1	18.9	19.7	14.9	24.3	18.9	17.1	15.2	23.4	18.8	16.0
7-MINMIN-MinMin	7.1	27.9	9.7	24.3	6.8	26.0	9.3	19.7	6.6	24.5	8.9	17.3	6.1	24.7	7.6	16.9
8-MinMin	7.4	42.3	10.0	17.7	6.7	40.8	8.9	12.6	6.5	39.6	8.7	9.9	6.4	38.9	7.7	8.7
9-MinMIN	15.8	11.5	11.2	14.4	14.5	9.5	7.0	9.5	13.3	8.3	2.6	7.2	13.0	7.8	2.4	6.3
10-MINMin	4.9	40.0	8.0	16.7	4.7	30.5	8.3	8.5	5.2	19.7	9.8	3.3	5.8	16.2	10.5	2.2
11-MINMIN	17.0	13.1	12.7	14.9	17.1	10.1	13.3	9.1	16.9	8.8	14.0	6.7	16.9	7.9	13.8	5.6
12-MaxMin-MINMin	2.6	5.5	4.9	11.6	2.1	3.8	5.5	8.0	2.1	2.9	8.0	6.4	2.0	2.1	9.2	5.8
13-MaxMin	**1.0**	**0.0**	3.0	9.3	**1.0**	**0.5**	3.3	6.9	**1.0**	**0.0**	5.5	5.6	**1.0**	**0.0**	8.3	5.6
14-MaxMIN	14.9	35.1	**1.5**	2.2	17.6	44.3	**1.3**	**0.9**	18.0	47.2	**1.7**	**0.4**	18.0	48.0	**2.2**	**1.0**
15-*kpb*.SuffMIN	19.8	54.1	17.8	24.9	19.9	60.9	19.0	21.0	20.0	62.8	17.6	16.7	19.9	64.3	16.6	14.0
16-*kpb*.Sufferage	10.0	19.0	12.3	16.6	8.4	13.0	9.0	9.7	7.8	9.8	8.9	6.7	7.4	8.4	8.7	5.8
17-*kpb*.MINSuff	11.6	24.1	15.1	19.7	11.4	17.4	14.7	12.4	11.3	14.2	14.8	9.3	11.4	12.8	14.7	8.3
18-*kpb*.MinMIN	11.8	49.1	15.4	27.3	11.6	36.7	14.7	14.6	11.2	24.5	14.4	7.3	11.2	20.0	13.2	5.2
19-*kpb*.MinMin	18.7	24.5	19.2	20.0	16.1	17.6	15.7	12.3	15.0	13.9	9.7	8.9	14.2	12.2	7.2	7.6
20-*kpb*.MaxMin	8.5	16.0	9.4	14.8	9.0	14.0	12.8	11.5	9.4	11.8	15.0	9.8	9.5	10.6	16.1	9.4
χ^2	3694.6		3415.1		3643.8		3068.7		3633.4		2814.1		3634.6		2680.3	
p-value	$<10^{-3}$		$<10^{-3}$		$<10^{-3}$		$<10^{-3}$		$<10^{-3}$		$<10^{-3}$		$<10^{-3}$		$<10^{-3}$	

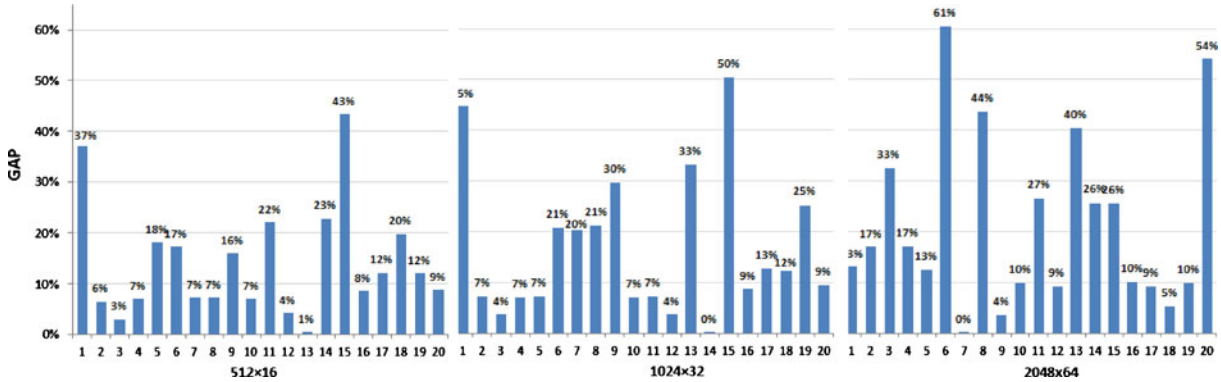


Fig. 5 Comparison of the average GAPs for the makespan objective. Labels 1 to 20 identify the different algorithms, as shown in Table 4

in the previous subsections (Sufferage, SuffMIN, MaxMin, MaxMin-MINMin, MaxMIN, and MINSuff) to solve ME-HCSP instances with variable ratios between the number of tasks and machines. The instances solved include scenarios with 16, 32, and 64 machines, and workloads with 1024, 2048, 4096, and 8192 tasks, with with low/medium/high heterogeneity.

Table 5 summarizes the average GAPs in the makespan and energy objectives for each heuristic and problem dimension. As in the previous subsection, the GAPs are computed with respect to the best value obtained for each objective for each instance.

The results reported in Table 5 indicate that the studied heuristics have a different behavior when the tasks/machines ratio varies. Regarding the makespan metric, MaxMin is the best heuristic for low tasks/machines ratios (16, 32, and 64), but the combined MINSuff heuristic and the traditional Sufferage obtained the best makespan values for larger ratios (128, 256, and 512). Regarding the energy consumption, MaxMIN is the best heuristic, and SuffMIN obtained accurate energy values, especially for instances with a moderate number of tasks per machine. Fig. 8 present a comparison of the four best heuristics regarding the makespan and energy objectives.

6.3.3 Execution Time Analysis

Table 6 reports the average execution time (in seconds) for each problem dimension and heuristic class (traditional, combined, alternated, greedy

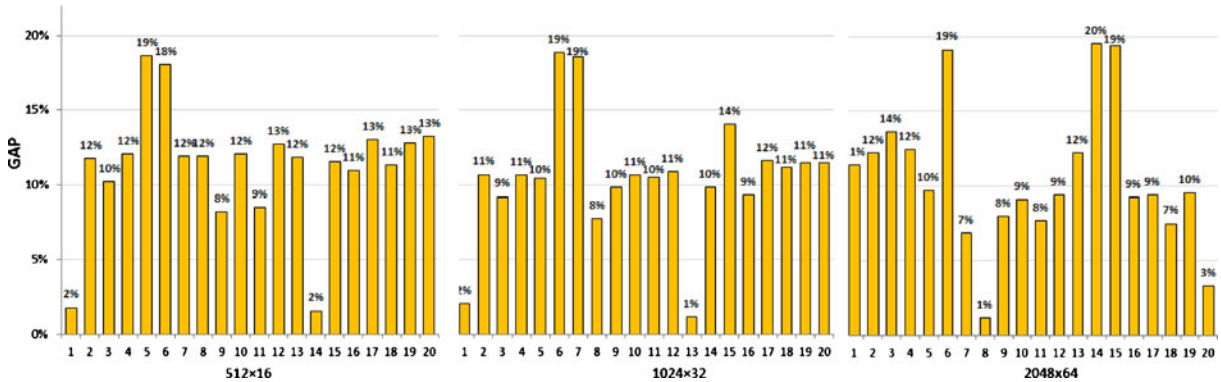
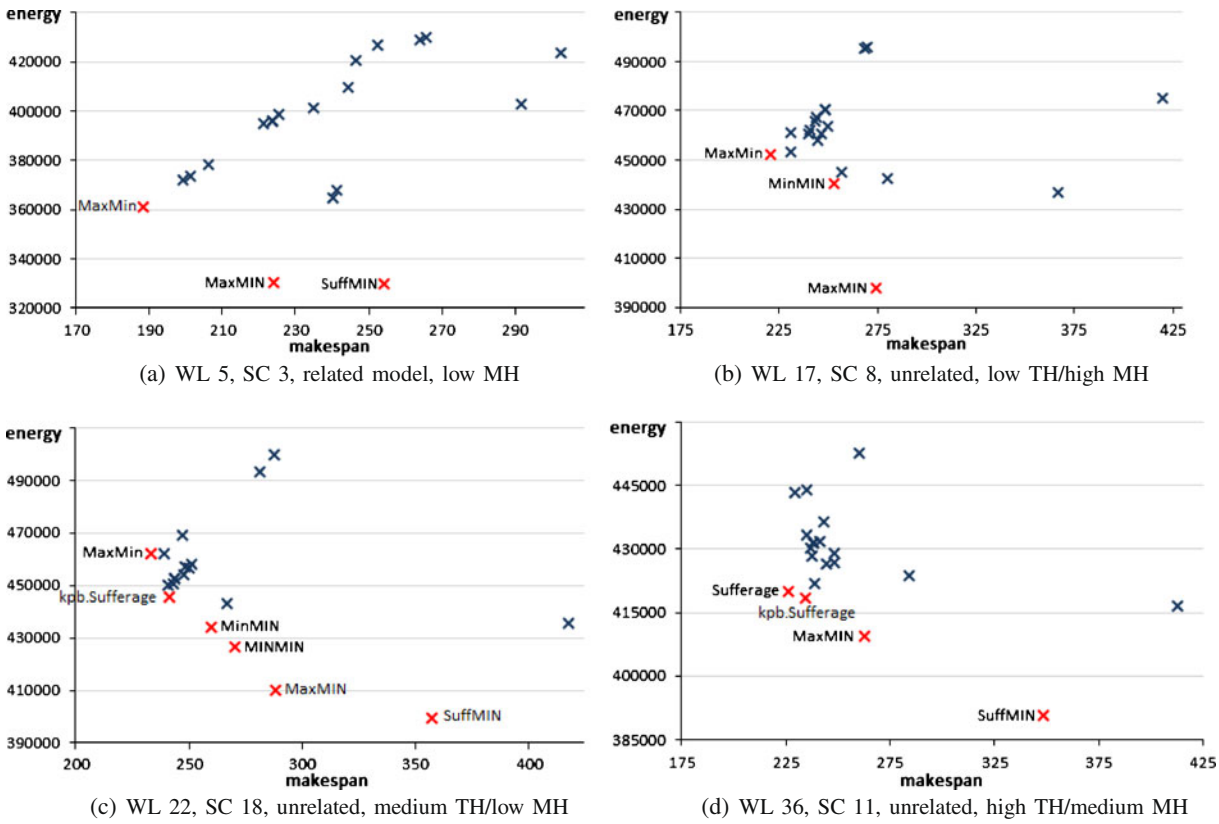


Fig. 6 Comparison of the average GAPs for the energy objective. Labels 1 to 20 identify the different algorithms, as shown in Table 4

Table 4 Dominance analysis for the proposed heuristics (512 × 16, 1,024 × 32 and 2,048 × 64 ME-HCSP instances)

	512 × 16					1,024 × 32					2,048 × 64				
	Related	Unrelated			Avg	Related	Unrelated			Avg	Related	Unrelated			Avg
		Low	Med.	High			Low	Med.	High			Low	Med.	High	
1-SuffMIN	4.68	0.97	0.87	0.79	1.83	2.5	1.0	0.9	0.9	1.3	2.1	1.0	1.0	0.9	1.2
2-Sufferage-MINMin	10.67	7.21	5.85	5.66	7.35	10.3	6.8	5.3	5.9	7.1	11.8	8.7	8.3	8.1	9.2
3-Sufferage	13.14	**12.17**	**11.25**	**10.30**	**11.71**	13.5	11.7	**11.2**	**10.4**	**11.7**	15.2	13.7	12.6	**12.6**	13.5
4-MINSuff	7.21	6.69	6.04	6.03	6.49	8.1	7.9	6.4	6.3	7.2	10.9	10.5	8.8	7.9	9.5
5-MinMIN-MINMin	1.60	1.00	0.65	0.56	0.95	1.3	0.9	0.8	0.7	0.9	2.1	1.2	1.1	0.7	1.3
6-MinMIN-MinMin	1.55	1.13	1.02	0.96	1.16	1.0	1.0	0.8	0.9	0.9	1.7	1.1	1.2	1.2	1.3
7-MINMIN-MinMin	6.61	6.70	7.54	7.63	7.12	7.9	8.0	7.8	8.6	8.1	10.0	10.0	9.5	10.0	9.9
8-MinMin	6.51	6.29	7.50	7.87	7.04	7.8	7.7	7.6	7.9	7.7	9.6	10.2	9.7	9.8	9.8
9-MinMIN	4.08	4.89	4.75	4.90	4.65	3.8	4.3	4.3	4.9	4.3	3.0	3.7	5.4	5.9	4.5
10-MINMin	7.21	6.69	6.04	6.03	6.49	8.1	7.9	6.4	6.3	7.2	10.9	10.5	8.8	7.9	9.5
11-MINMIN	3.55	2.22	1.74	1.66	2.29	3.0	1.7	1.5	1.5	1.9	2.1	1.2	1.1	1.1	1.4
12-MaxMin-MINMin	12.19	7.13	3.31	2.79	6.35	13.6	9.3	4.4	3.8	7.8	15.1	14.5	11.9	10.6	13.0
13-MaxMin	**14.85**	11.77	4.73	3.43	8.69	**16.4**	**12.9**	6.6	4.8	10.2	**17.0**	**16.7**	**14.5**	11.7	**15.0**
14-MaxMIN	7.60	3.94	2.32	1.75	3.90	5.5	2.3	1.8	1.6	2.8	4.7	2.2	1.8	1.7	2.6
15-*kpb*.SuffMIN	1.02	0.00	0.01	0.03	0.26	0.5	0.0	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0
16-*kpb*.Sufferage	6.19	7.70	8.01	7.88	7.44	6.5	8.6	9.5	9.6	8.6	7.1	8.7	8.8	8.9	8.4
17-*kpb*.MINSuff	2.53	2.89	3.35	3.98	3.19	3.9	4.3	3.6	3.9	3.9	4.7	4.9	4.3	4.1	4.5
18-*kpb*.MinMIN	0.32	1.83	2.62	3.02	1.95	0.3	1.9	2.7	3.1	2.0	4.4	4.8	4.5	4.6	4.6
19-*kpb*.MinMin	2.30	3.15	3.88	4.80	3.53	3.6	4.5	4.6	5.1	4.5	0.3	1.3	2.9	3.8	2.1
20-*kpb*.MaxMin	8.27	5.80	3.01	2.11	4.79	9.4	5.5	3.5	2.8	5.3	8.9	6.7	5.0	3.9	6.1



(a) WL 5, SC 3, related model, low MH

(b) WL 17, SC 8, unrelated, low TH/high MH

(c) WL 22, SC 18, unrelated, medium TH/low MH

(d) WL 36, SC 11, unrelated, high TH/medium MH

Fig. 7 Four examples of makespan/energy results for different workloads and scenarios (512 × 16 ME-HCSP instances)

energy). The heuristics have been grouped this way since the algorithmic skeleton is the same for all methods in each class, thus similar execution times are expected within each category. We distinguish between *kpb traditional* and *kpb combined* since the execution times varies when the *kpb* schema is applied over a traditional or a combined heuristic.

The execution time values in Table 6 state that the traditional heuristics are the fastest ones, since they only compute the energy consumption once, after the schedule has been found. Combined heuristics are on average 30 % slower than the traditional methods, mainly because they need to compute the energy consumption in every loop step, and alternated heuristics lay in between traditional and combined ones.

The greedy energy MINMIN heuristic demands large computing times, since the energy consumption evaluation is performed into both the inner and outer phases. The results also show that the *kpb* strategy is an efficient way of reducing the execution times of the base method; the execution times are reduced on 30 % for the traditional and about 20 % for the combined heuristics.

The previously presented execution times should be taken into account, along with the makespan and energy consumption results, when searching for accurate and efficient schedulers, especially when large workloads are submitted to large HC infrastructures and Grid systems.

6.4 Summary of Results

As it has been shown in Section 6.3, the twenty studied heuristics look for different trade-off solutions to the ME-HCSP, and their accuracy depends on the instance size, the task/machines ratio, and/or heterogeneity.

In order to extract some global conclusions on the behavior of the algorithms, and to summarize our findings, we ranked the heuristics according to

Table 5 Scalability analysis for different tasks/machines ratios

Machines	1,024			2,048			4,096			8,192		
Tasks	16	32	64	16	32	64	16	32	64	16	32	64
Ratio	(64)	(32)	(16)	(128)	(64)	(32)	(256)	(128)	(64)	(512)	(256)	(128)
	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)
Makespan objective												
1-SuffMIN	46.2	30.6	44.0	47.7	49.7	55.4	45.4	47.8	53.5	46.8	52.2	55.8
3-Sufferage	1.0	3.2	3.8	0.5	1.1	5.9	0.3	0.6	2.2	0.4	0.3	0.6
4-MINSuff	4.3	8.9	8.2	1.2	2.0	7.5	0.3	0.9	2.2	0.1	0.2	0.7
12-MaxMin-MINMin	2.5	4.4	0.4	2.9	1.3	3.7	2.9	2.3	1.6	5.0	4.0	2.5
13-MaxMin	1.4	0.0	0.1	2.9	0.2	0.0	3.7	2.6	0.0	7.0	5.2	2.6
14-MaxMIN	47.2	16.3	43.7	50.2	51.2	48.5	52.0	53.7	46.1	52.5	52.0	53.5
Energy objective												
1-SuffMIN	3.4	0.0	4.7	6.0	0.8	1.7	3.5	0.0	5.6	4.1	2.9	2.7
3-Sufferage	8.1	11.0	9.9	7.4	5.0	8.7	5.1	4.7	7.4	5.3	3.1	2.8
4-MINSuff	9.4	13.8	12.0	7.6	5.4	9.4	5.0	4.8	7.5	5.0	3.1	2.8
12-MaxMin-MINMin	9.9	13.1	10.0	9.9	6.3	9.1	7.7	6.7	8.2	9.8	7.0	5.4
13-MaxMin	10.2	11.3	9.9	10.5	6.2	8.1	8.7	7.4	7.8	11.9	8.3	5.8
14-MaxMIN	0.4	2.1	0.0	0.0	1.5	1.5	0.2	2.2	0.0	0.0	0.7	0.0

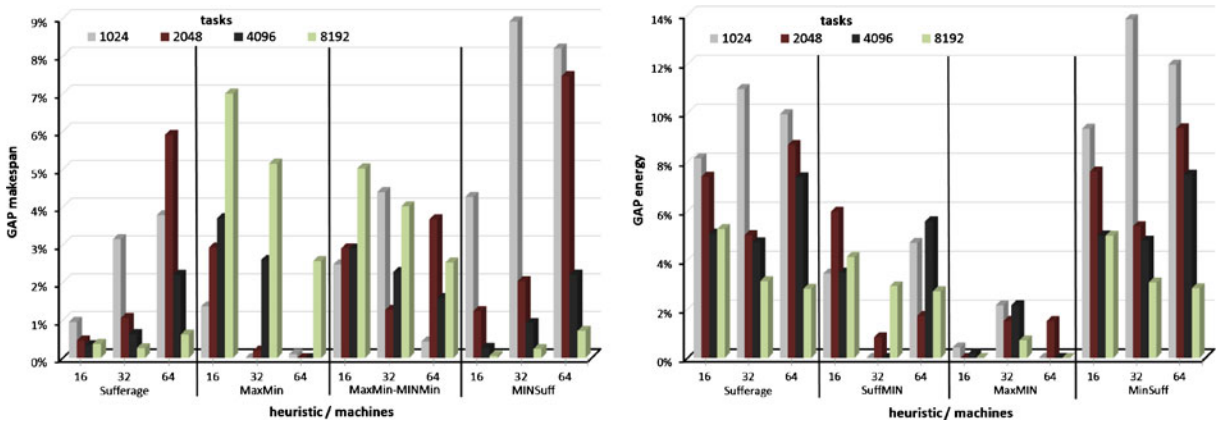
their overall performance on all related and unrelated instances, for the three considered instance sizes, in Table 7. For that, we averaged the rank values provided in Table 3 (these mean rank values are available in Table 7). The table shows the best algorithms according to makespan, energy, and both of them (the overall column).

The classic heuristics from the literature find highly balanced solutions, because they are focussed on makespan minimization. The schedules they find have very low processor idle times, therefore minimizing the energy waste. It can be seen in Table 3 that MaxMin, the best heuristic according to makespan, requires between 5.3 % (for the largest intances) to 13.1 % (for the smallest ones) higher energy consumption than the most energy efficient one. On the contrary, MaxMIN, which finds the most energy efficient schedules, shows difficulties to find load balanced solutions, as it is demonstrated by the obtained makespan values, ranging between 20.7 % (for 512×16 instances) and 48 % (for $2{,}048 \times 64$ instances) worse than the best.

The best overall heuristic according to the two objectives was Sufferage, being MaxMin very close to it (although Table 7 shows the same rank values for they two because of rounding). Sufferage was between 1.7% (for the small instance) to 6.1 % (for the large problems) worse than the best result in terms of makespan, and between 4.8 and 12.1 % worse than the most energy-efficient results.

The alternated heuristics seem an appropriate option to find accurate solutions in terms of the two studied objectives. Among them, MaxMin-MINMin is selected as the best performing one (it is the third best heuristic overall), with results ranging between 2.1 and 5.9 % worse than the best makespan found and consuming between 5.8 and 13.3 % more than the most efficient solution. Additionally, it showed a competitive performance on the scalability and execution time analysis. It outperformed MaxMin both in terms of makespan and energy for the largest instances, and it is generally better than Sufferage in terms of makespan for those instances with higher number of tasks.

Regarding the *kpb* heuristics, they could be a good option in case a quick response is needed in the system, since they are around 20 % faster



(a) makespan objective (b) energy objective

Fig. 8 Scalability analysis for different tasks/machines ratios

Table 6 Average execution times (in seconds) for each heuristic class and problem dimension

Dimension	Heuristic class					
	Traditional	Combined	Alternated	Greedy energy	kpb traditional	kpb combined
512 × 16	0.02	0.03	0.03	0.21	0.02	0.02
1,024 × 32	0.09	0.11	0.09	1.45	0.06	0.11
2,048 × 64	0.54	0.61	0.47	11.06	0.32	0.53
4,096 × 64	1.87	2.45	1.86	45.17	1.22	2.15
8,192 × 64	6.93	9.60	7.39	181.10	4.43	9.15

Table 7 Final rank of the proposed heuristics on all instances

Position	Makespan		Energy		Overall	
	Algorithm	Rank	Algorithm	Rank	Algorithm	Rank
1	MaxMin	1.2	MaxMIN	1.7	Sufferage	4.9
2	MaxMin-MINMin	2.9	SuffMIN	2.7	MaxMin	4.9
3	Sufferage	2.9	MinMIN	5.4	MaxMin-MINMin	6.9
4	MINMin	5.6	Sufferage	6.9	MINMin	8.0
5	MINMIN-MinMin	6.6	*kpb*.Sufferage	8.7	MINMIN-MinMin	8.3
6	MINSuff	6.6	MaxMin	8.7	*kpb*.Sufferage	8.4
7	Sufferage-MINMin	6.7	MINMIN	9.3	MinMin	8.4
8	MinMin	6.7	MINMIN-MinMin	10.1	Sufferage-MINMin	9.0
9	*kpb*.Sufferage	8.0	MinMin	10.2	MINSuff	9.0
10	*kpb*.MaxMin	8.9	MINMin	10.3	MinMIN	9.5
11	*kpb*.MINSuff	11.7	MaxMin-MINMin	10.8	MaxMIN	9.7
12	*kpb*.MinMin	13.0	Sufferage-MINMin	11.1	SuffMIN	10.6
13	MinMIN	13.7	MINSuff	11.3	*kpb*.MaxMin	11.1
14	*kpb*.MinMIN	14.3	*kpb*.MinMIN	11.5	*kpb*.MINSuff	12.8
15	MinMIN-MinMin	14.9	*kpb*.MinMin	12.5	*kpb*.MinMin	12.8
16	MinMIN-MINMin	15.0	*kpb*.MaxMin	13.3	MINMIN	12.9
17	MINMIN	16.5	*kpb*.MINSuff	13.8	*kpb*.MinMIN	12.9
18	MaxMIN	16.5	*kpb*.SuffMIN	14.1	MinMIN-MinMin	16.8
19	SuffMIN	18.5	MinMIN-MinMin	18.8	MinMIN-MINMin	16.9
20	*kpb*.SuffMIN	19.8	MinMIN-MINMin	18.9	*kpb*.SuffMIN	16.9

than the classic heuristics, although they are not so accurate.

7 Conclusions and Future Work

This work introduced twenty novel list scheduling heuristics for the simultaneous optimization of makespan and energy consumption in HC systems. These heuristics are argued to be more appropriate than other existing aggregative and Pareto approaches for the considered problem. A mathematical model for the energy consumption in multicore HC Grid systems has been formulated. The resulting model has been used to conceive fast scheduling heuristics, following a simple methodology that focuses on the two-phases structure of the traditional list scheduling process. By applying diverse combinations of makespan and energy optimization, twenty new bi-objective scheduling list heuristics have been proposed.

The new heuristics have been studied in an exhaustive experimental analysis performed on a large set of ME-HCSP instances. This new set of problem instances has been designed following well-known methodologies for scenarios with both the related and unrelated machines model. In addition, computing power and energy consumption values were obtained for 64 different modern processors, using data from the SSJ energy efficiency benchmark, from Standard Performance Evaluation Corporation. The problem instances defined allow modeling realistic HC and Grid infrastructures. The experimental analysis included the evaluation of the proposed heuristics using large instances –up to 2048 tasks, 64 machines, and 315 cores in the evaluation, and up to 8192 tasks in the scalability analysis–, and 800 instances with different workload/machine heterogeneity were used for each dimension studied.

The experimental evaluation allowed to identify some promising heuristics for the bi-objective ME-HCSP. Regarding the makespan optimization, the traditional MaxMin heuristic was consistently the best method, and Sufferage was the second-best option. When considering the energy consumption objective, combined heuristics such as SuffMIN, MaxMIN, and MinMIN, and also some *kpb* variants, obtained the best energy values.

The proposed heuristics compute solutions with different trade-offs between makespan and energy. In the Pareto-dominance analysis, Sufferage, MaxMin, and some alternated variants as MaxMin-MINMin had the largest number of dominating solutions overall. Both a scalability analysis and a execution time study were made. We conclude that the best overall heuristic is MaxMin-MINMin, especially when dealing with large problem sizes.

All the previously presented results have direct application in modern schedulers. The best new heuristics are able to compute accurate solutions regarding the two objectives considered, requiring a very short execution time, even for the largest problem instances studied. In this sense, MaxMin-MINMin was even able to outperform MaxMin in terms of the two objectives for the largest instances studied in the scalability test.

The resulting schedules can be applied directly, or they can be used to seed more powerful optimization methods that intend to further improve the assignment. The diversity provided by the different list scheduling variants is also useful when different trade-offs are considered in diverse HC and Grid infrastructures.

The main lines for future work are related with further improving the search capabilities of the proposed methods, possibly by including a fast and powerful local search that explores possible move-and-swap tasks assignments, regarding one or both problem objectives simultaneously. Additionally, we plan to extend our work with the scheduling of workflows. Scheduling in Grid computing, where time sharing of tasks on machines is available, providing dynamic variation of their speed, is an important issue to be addressed. Moreover, the communication latency is a relevant issue to include in the model.

Acknowledgements The work of S. Nesmachnow has been partially supported by ANII and PEDECIBA, Uruguay. B. Dorronsoro acknowledges the support by the Fonds National de la Recherche, Luxembourg (AFR contract no 4017742). The work of J. E. Pecero, and P. Bouvry is supported by the Fonds National de la Recherche Luxembourg, CORE Project Green-IT (C09/IS/05).

References

1. Condor high throughput computing: Available online at http://www.cs.wisc.edu/condor/ (2011). Cited August 2011
2. Globus: Available online at http://www.globus.org/ (2012). Cited November 2012.
3. Openpbs: Available online at http://www.mcs.anl.gov/research/projects/openpbs/ (2012). Cited November 2012
4. Lsf—load sharing facility: Available online at http://www.vub.ac.be/BFUCC/LSF/ (2013). Cited January 2013.
5. Oar resource management system for high performance computing: Available online at http://oar.imag.fr (2013). Cited January 2013
6. Ali, S., Siegel, H.J., Maheswaran, M., Ali, S., Hensgen, D.: Task execution time modeling for heterogeneous computing systems. In: Proc. of the 9th Heterogeneous Computing Workshop, p. 185. IEEE Computer Society, Washington DC (2000)
7. Ambrosio, P., Auletta, V.: Deterministic monotone algorithms for scheduling on related machines. Theory Comput. Sci. **406**, 173–186 (2008)
8. Barrondo, A., Tchernykh, A., Schaeffer, E., Pecero, J.E.: Energy efficiency of knowledge-free scheduling in peer-to-peer desktop Grids. In: Proc. of the International Conference on High Performance Computing & Simulation (HPCS), pp. 105–111 (2012)
9. Baskiyar, S., Abdel-Kader, R.: Energy aware DAG scheduling on heterogeneous systems. Cluster Comput. **13**, 373–383 (2010)
10. Baskiyar, S., Palli, K.: Low power scheduling of DAGs to minimize finish times. In: Robert, Y., Parashar, M., Badrinath, R., Prasanna, V. (eds.) High Performance Computing—HiPC 2006. Lecture Notes in Computer Science, vol. 4297, pp. 353–362. Springer Berlin, Heidelberg (2006)
11. Berman, F., Fox, G., Hey, A.: Grid Computing: Making the Global Infrastructure a Reality. Wiley, New York (2003)
12. Blazewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., Weglarz, J.: Handbook on Scheduling: From Theory to Applications. Springer (2007)
13. Braun, T., Siegel, H.J., 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 onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **61**(6), 810–837 (2001)
14. Buscemi, M.G., Montanari, U., Taneja, S.: A game-theoretic analysis of Grid job scheduling. J. Grid Computing **10**(3), 501–519 (2012)
15. Coutinho, F., de Carvalho, L.A.V.: Strategies based on green policies to the Grid resource allocation. In: Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques, PACT '12, pp. 487–488. ACM, New York (2012)
16. Da Costa, G., Gelas, J.-P., Georgiou, Y., Lefevre, L., Orgerie, A.-C., Pierson, J.-M., Richard, O., Sharma, K.: The green-net framework: Energy efficiency in large scale distributed systems. In: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing, IPDPS '09, pp. 1–8. IEEE Computer Society, Washington (2009)
17. Dail, H., Sievert, O., Berman, F., Casanova, H., YarKhan, A., Vadhiyar, S., Dongarra, J., Liu, C., Yang, L., Angulo, D., Foster, I.: Grid Resource Management, chapter Scheduling in the Grid Application Development Software Project, pp. 73–98. Kluwer Academic Publishers (2004)
18. Develder, C., Pickavet, M., Dhoedt, B., Demeester, P.: A power-saving strategy for Grids. In: Proceedings of GridNets2008, the ICST 2nd International Conference on Networks for Grid Applications and Workshops. ICST (2008)
19. Dorronsoro, B., Bouvry, P., Cañero, J., Maciejewski, A., Siegel, H.: Multi-objective robust static mapping of independent tasks on Grids. In: IEEE Congress on Evolutionary Computation (CEC). Part of the World Congress on Computational Intelligence (WCCI), pp. 3389–3396 (2010)
20. El-Rewini, H., Lewis, T., Ali, H.: Task Scheduling in Parallel and Distributed Systems. Prentice-Hall Inc. (1994)
21. Eshaghian, M.: Heterogeneous Computing. Artech House (1996)
22. Eyraud, L.: A pragmatic analysis of scheduling environments on new computing platforms. Int. J. High Perform. Comput. Appl. **20**(4), 507–516 (2006)
23. Foster, I., Kesselman, C.: The Grid 2: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers (2003)
24. Freund, R., Gherrity, M., Ambrosius, S., Campbell, M., Halderman, M., Hensgen, D., Keith, E., Kidd, T., Kussow, M., Lima, J., Mirabile, F., Moore, L., Rust, B., Siegel, H.: Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In: Proc. of the 7th Heterogeneous Computing Workshop, p. 3. IEEE Computer Society, Washington DC (1998)
25. Freund, R., Sunderam, V., Gottlieb, A., Hwang, K., Sahni, S.: Special issue on heterogeneous processing. J. Parallel Distrib. Comput. **21**(3) (1994)
26. Garey, M., Johnson, D.: Computers and Intractability. Freeman (1979)
27. Garg, S.K., Buyya, R., Siegel, H.J.: Scheduling parallel applications on utility Grids: time and cost trade-off management. In: Proc. of the 32nd Australasian Computer Science Conference ACSC, vol. 91, pp. 139–147 (2009)
28. Kumar Garg, S., Buyya, R.: Exploiting heterogeneity in Grid computing for energy-efficient resource allocation. In: Proceedings of the 2009 17th International Conference on Advanced Computing and Communications, ADCOM 2009, pp. 1–7. Bengaluru, India (2009)
29. GISELA: Grid Infrastructures for e-Science virtual communities in Europe and Latin-America (GISELA) project. Available at http://www.gisela-grid.eu/ (2011). Retrieved April 2011.

30. Gray, L., Kumar, A., Li, H.: Workload Characterization of the SPECpower_ssj2008 Benchmark. In: Kounev, S., Gorton, I., Sachs, K. (eds.) Performance Evaluation: Metrics, Models and Benchmarks. Lecture Notes in Computer Science, vol. 5119, pp. 262–282. Springer Berlin/Heidelberg (2008)
31. Hermenier, F., Loriant, N., Menaud, J.-M.: Power management in Grid computing with xen. In: Proceedings of the 2006 International Conference on Frontiers of High Performance Computing and Networking, ISPA'06, pp. 407–416. Springer, Berlin, Heidelberg (2006)
32. Hirales-Carbajal, A., Tchernykh, A., Yahyapour, R., González-García, J.L., Röblitz, T., Ramírez-Alcaraz, J.M.: Multiple workflow scheduling strategies with user run time estimates on a Grid. J. Grid Computing **10**(2), 325–346 (2012)
33. Ibarra, O., Kim, C.: Heuristic algorithms for scheduling independent tasks on nonidentical processors. J. ACM **24**(2), 280–289 (1977)
34. Kertész, A., Kacsuk, P.: Gmbs: A new middleware service for making Grids interoperable. Future Gener. Comput. Syst. **26**(4), 542–553 (2010)
35. 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. **20**, 346–360 (2009)
36. Kim, J.-K., Siegel, H.J., Maciejewski, A., Eigenmann, R.: Dynamic resource management in energy constrained heterogeneous computing systems using voltage scaling. IEEE Trans. Parallel Distrib. Syst. **19**, 1445–1457 (2008)
37. Kim, K., Buyya, R., Kim, J.: Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters. In: Proc. of the 7th IEEE International Symposium on Cluster Computing and the Grid, pp. 541–548. IEEE Computer Society, Washington DC (2007)
38. Kolodziej, J., Xhafa, F.: Intelligent Decision Systems in Large-Scale Distributed Environments. Studies in Computational Intelligence, chapter Task Allocation Oriented Users Decisions in Computational Grid, vol. 362, pp. 1–23. Springer Berlin-Heidelberg (2011)
39. Kurowski, K., Nabrzyski, J., Oleksiak, A., Weglarz, J.: A multicriteria approach to two-level hierarchy scheduling in Grids. J. Scheduling **11**(5), 371–379 (2008)
40. Kurowski, K., Oleksiak, A., Witkowski, M., Nabrzyski, J.: Distributed power management and control system for sustainable computing environments. In: International Green Computing Conference, pp. 365–372. IEEE (2010)
41. Kwok, Y., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. **31**(4), 406–471 (1999)
42. Lammie, M., Brenner, P., Thain, D.: Scheduling Grid workloads on multicore clusters to minimize energy and maximize performance. In: GRID, pp. 145–152. IEEE (2009)
43. Lee, Y.C., Zomaya, A.: Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In: Proc. of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 92–99. IEEE Computer Society, Washington DC (2009)
44. Lee, Y.C., Zomaya, A.: Energy conscious scheduling for distributed computing systems under different operating conditions. IEEE Trans. Parallel Distrib. Syst. **22**, 1374–1381 (2011)
45. Lenstra, J., Shmoys, D., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. **46**, 259–271 (1990)
46. Leung, J., Kelly, L., Anderson, J.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Inc. (2004)
47. Li, K.: Optimal load distribution for multiple heterogeneous blade servers in a cloud computing environment. J. Grid Comput. **11**(1), 27–46 (2013)
48. Li, Y., Liu, Y., Qian, D.: A heuristic energy-aware scheduling algorithm for heterogeneous clusters. In: Proc. of the 2009 15th International Conference on Parallel and Distributed Systems, ICPADS '09, pp. 407–413. IEEE Computer Society, Washington DC (2009)
49. 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. J. Supercomputing **59**(1), 323–360 (2012)
50. Ludwig, S.A., Moallem, A.: Swarm intelligence approaches for Grid load balancing. J. Grid Computing **9**(3), 279–301 (2011)
51. Luo, P., Lü, K., Shi, Z.: A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems. J. Parallel Distrib. Comput. **67**(6), 695–714 (2007)
52. 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. **71**, 1497–1508 (2011)
53. Minas, L., Ellison, B.: Energy Efficiency for Information Technology: How to Reduce Power Consumption in Servers and Data Centers. Intel Press (2009)
54. Nesmachnow, S.: Cluster FING: High Performance Computing at Universidad de la República. Available online at http://www.fing.edu.uy/cluster (2011). Accessed June 2011
55. Orgerie, A.-C., Lefèvre, L., Gelas, J.-P.: Chasing gaps between bursts: towards energy efficient large scale experimental Grids. In: Proceedings of the 2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT '08, pp. 381–389. IEEE Computer Society, Washington DC (2008)
56. Orgerie, A.-C., Lefèvre, L., Gelas, J.-P.: Save watts in your Grid: green strategies for energy-aware framework in large scale distributed systems. In: Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems, ICPADS '08, pp. 171–178. IEEE Computer Society, Washington, DC (2008)

57. Pecero, J., Bouvry, P., Fraire Huacuja, H.J., Khan, S.: A multi-objective grasp algorithm for joint optimization of energy consumption and schedule length of precedence-constrained applications. In: Int. Conf. Cloud and Green Computing, pp. 1–8. IEEE CS Press, Sydney (2011)
58. Pinel, F., Dorronsoro, B., Pecero, J.E., Bouvry, P., Khan, S.U.: A two-phase heuristic for the energy-efficient scheduling of independent tasks on computational Grids. Clust. Comput. (2012). doi:10.1007/s10586-012-0207-x
59. Pugliese, A., Talia, D., Yahyapour, R.: Modeling and supporting Grid scheduling. J. Grid Computing **6**, 195–213 (2008)
60. Ramírez-Alcaraz, J.M., Tchernykh, A., Yahyapour, R., Schwiegelshohn, U., Quezada-Pina, A., González-García, J.L., Hirales-Carbajal, A.: Job allocation strategies with user run time estimates for online scheduling in hierarchical Grids. J. Grid Computing **9**(1), 95–116 (2011)
61. Riedel, M., Laure, E., et al.: Interoperation of worldwide production e-science infrastructures. Concurr. Comput.: Pract. Exper. **21**(8), 961–990 (2009)
62. Rizvandi, N., Taheri, J., Zomaya, A.: Some observations on optimal frequency selection in dvfs-based energy consumption minimization. J. Parallel Distrib. Comput. **71**, 1154–1164 (2011)
63. Rodero, I., Guim, F., Corbalán, J., Fong, L., Sadjadi, S.M.: Grid broker selection strategies using aggregated resource information. Future Gener. Comput. Syst. **26**(1), 72–86 (2010)
64. Schwiegelshohn, U., Yahyapour, R.: Attributes for communication between Grid scheduling instances. In: Nabrzyski, J., Schopf, J.M., Weglarz, J. (eds.) Grid Resource Management. International Series in Operations Research & Management Science, vol. 64, pp. 41–52. Springer US, Norwell (2004)
65. Shmoys, D.B., Wein, J., Williamson, D.P.: Scheduling parallel machines on-line. SIAM J. Comput. **24**(6), 1313–1331 (1995)
66. SPEC: Standard performance evaluation corporation, power and performance methodology. Available online at http://www.spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf (2011). Accessed September 2011
67. SPEC: Standard performance evaluation corporation, specpower_ssj2008. Available online at http://www.spec.org/power_ssj2008 (2011). Accessed September 2011
68. Subrata, R., Zomaya, A.Y., Landfeldt, B.: Cooperative power-aware scheduling in Grid computing environments. J. Parallel Distrib. Comput. **70**(2), 84–91 (2010)
69. Tchernykh, A., Schwiegelshohn, U., Yahyapour, R., Kuzjurin, N.: On-line hierarchical job scheduling on Grids with admissible allocation. J. Scheduling **13**(5), 545–552 (2010)
70. Xhafa, F., Abraham, A.: Computational models and heuristic methods for Grid scheduling problems. Future Gener. Comput. Syst. **26**, 608–621 (2010)
71. Yin, F., Jiang, C., Deng, R., Yuan, J.: Grid resource management policies for load-balancing and energy-saving by vacation queuing theory. Comput. Electr. Eng. **35**(6), 966–979 (2009)
72. Yu, J., Kirley, M., Buyya, R.: Multi-objective planning for workflow execution on Grids. In: IEEE/ACM Int. Conf. on Grid Computing, pp. 10–17 (2007)
73. 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. **14**, 686–700 (2003)
74. Zomaya, A., Teh, Y.: Observations on using genetic algorithms for dynamic load-balancing. IEEE Trans. Parallel Distrib. Syst. **12**, 899–911 (2001)