Methods for Job Scheduling on Computational Grids: Review and Comparison

Edson Flórez[1(✉)], Carlos J. Barrios[1], and Johnatan E. Pecero[2]

[1] High Performance and Scientific Computing Center - SC3UIS, Universidad Industrial de Santander, Bucaramanga, Colombia
edson.florez@correo.uis.edu.co, cbarrios@uis.edu.co
[2] Computer Science and Communications Research Unit, University of Luxembourg, Luxembourg City, Luxembourg
johnatan.pecero@gmail.com

Abstract. This paper provides a review of heuristics and metaheuristics methods, to solve the job scheduling problem in grid systems under the ETC (Expected Time to Compute) model. The problem is an important issue for efficient resource management in computational grids, which is performed by schedulers of these High Performance Computing systems. We present an overview of methods and a comparison of the results reported in the papers that use ETC model. The best methods are identified according to Braun et al. instances [8], which are ETC model instances most used in literature. This survey can help new researchers to lead them directly at the best scheduling algorithms already available to perform deep future works.

Keywords: High performance computing · Grid computing · Combinatorial optimization · Energy efficiency · Heuristics · Scheduling

1 Introduction

High performance computing systems such as grid computing are managed with a Resource Management System (RMS) [5], which typically has a resource manager and a task scheduler to guarantee both quality of service (QoS) provided to users and the requirements of the system administrator [4]. Therefore, an efficient scheduling algorithm is essential to fully exploit grid systems resources [10].

The purpose of this review article is analyze heuristic and metaheuristic approaches to solve job scheduling problem present in grid systems, according to the ETC model, which is a computation model for grid scheduling that allow to formalize, implement and evaluate different scheduling algorithms [20]. For this purpose, the job scheduling context in grid computing is introduced to provide a clear image of the optimization problem tackled, constraints and test cases. Furthermore, the most important heuristic and metaheuristic methods are briefly described.

Then, the analysis focuses on the works that reported their results with the most used ETC model instances, which were proposed by Braun et al. [8], in order to identify the best heuristics accurately. Comparison tables of performance and energy consumption of the reported results are provided. Finally, the outstanding job scheduling algorithms

C. Osthoff et al. (Eds.): CARLA 2015, CCIS 565, pp. 19–33, 2015.
DOI: 10.1007/978-3-319-26928-3_2

are investigated from different points of view, describing the distinctive characteristics of each algorithm, adaptation to the problem, strategies to face the complexity of the problem, etc. However, because a survey of an immense area such as grid scheduling has to focus on certain aspects and neglect others, the scope of the review reported in this article is only the ETC model, therefore, our selection of the best algorithms is only valid in this specific context, the results can be very different with other models and problems.

The paper is organized as follows. Section 2 presents an overview of scheduling problem in grid systems. We have summarized the main heuristics and metaheuristic methods for scheduling in Sect. 3, and we compare and analyze this methods in Sect. 4, delving into the most highlighted. Finally, Sect. 5 presents the main conclusions and lines for future work.

2 Overview of Job Scheduling Problem in Grid Systems

The need of high performance computing in today's society continues to increase and so does the datacenters size and their energy demand, which leads to a rise in the operating costs and carbon emissions. This paper seeks to identify efficient heuristics to allocate independent jobs in the resources of high performance computing systems, with the aiming to minimize energy consumption without affecting the system throughput.

Grid computing infrastructures are typically homogeneous, but the current trend is towards infrastructures with heterogeneous resources, that have nodes with different processing capacity and architectures to run applications or heterogeneous tasks, which gives them a great advantage over homogeneous platforms that deny service to users with different requirements. In heterogeneous infrastructures, job scheduling problem becomes even more complex and essential to the efficient use of resources.

Due to the high complexity of the job scheduling problem (NP-hard [2]) is not possible to determine the optimal solution, because the huge search space cannot be fully explored in reasonable time (polynomial) [18]. This requires approximation methods to find good quality solutions with high computational efficiency instead of algorithms that perform exhaustive searches that require a considerable computing time and resources. Among these are included heuristics and metaheuristic methods of artificial intelligence, such as the Genetic Algorithm (GA) [2].

2.1 Scheduling Problem in Grid Systems

Grid computing infrastructures may be focused on system or users. When is system-centred, the overall performance of the cluster is optimized as disk space, access range and energy consumption, so don't satisfy the users requirements. Instead, when is user-centred, criteria related utility achieved by users are evaluated, such as cost and execution time of their tasks, which can degrade system performance. The balance point (which covers the interests of the user and system administrator) is maximize the resource utilization holding good response times for users requests.

Performance metric most studied in the literature of grid scheduling is the makespan [3, 38], its minimization is a system administrator requirement because it allows to maximize resource usage, while improves the quality of service provided to the user, by reducing of the flowtime or length of the time interval between the release time and completion time of the task [9], which is related to the response times. The makespan is the *maximum completion time* C_i of the tasks [16, 17], is the time when the last task is finished, i.e., the ending time (or length) of the schedule [6]. Moreover, the energy efficiency is a very important issue for system administrator, established as performance per power consumption, to reduce high costs of operating the computing infrastructure, caused by high energy consumption of its multiple processors and the necessity of cooling to dissipate the heat produced.

Types of Job Scheduling. A job is defined as a monolithic application (i.e., a single task) or a collection of tasks, which can be dependent or independent [14]. Tasks are indivisible computational workload units, if they are dependent tasks, have to accomplish a predefined order dependencies between them, and if they are independent tasks, task doesn't require any communication with others [39].

Resources of HPC infrastructures are managed with Resource Management System (RMS), which defines how the computing resources are used to execute the user's tasks, by a dynamic or static scheduling. The dynamic scheduling takes into account the changing resources state, so schedule is modified on-line [11], by adding or removing processors to the jobs. In static scheduling is considered that the resources are available during a time interval (between two successive activations scheduler) and all jobs are known a priori [11], so the schedule is produced off-line, i.e., before starting its execution (and is not modified during execution).

Similarly, job processing can be batch type or immediate [20]; in immediate scheduling or on-line, jobs are assigned individually regardless the tasks that arriving after to the system, without resource planning. Instead, batch scheduling has a compute phase of tasks schedule, where these are grouped into batches, and then at the assign phase, tasks are allocated to selected resources according to the planning of the scheduler [20]. Due to jobs are assigned in groups, each job must wait for the next time interval when the scheduler get activated again. An immediate scheduling algorithm is most appropriate for clusters with low concurrence (which have good availability of resources), i.e., the job arrival rate is small having thus available resources to execute jobs immediately [20], at the time they are submitted by user, without any waiting time in queue. However, batch scheduling can take advantage of job information and compatible resources characteristics, for resource planning where determines which job is most suited to allocate to each resource, thus resulting in better makespan than immediate scheduling [6]. We considered static job scheduling type batch with instances produced under the ETC model [31].

2.2 ETC Computational Model

ETC is a computational model for grid scheduling, with which problem instances are easily represented, helping to the implementation and evaluation of scheduling

algorithms. It is based on a normal distribution or gamma to produce ETC matrices that contain estimated processing time of each task in each machine of the system [31].

Formal Definition of the Problem. A computational grid composed by a set of t independent tasks to be assigned and m machines available for planning, with an execution time ETC[t_i][m_j] previously known [2], which represents tasks and machines characteristics such as the computing capacity of machines and the workload of tasks. The workload of real applications can be obtained from specifications provided by the user, historical data or predictions [20].

$$\textit{Objective function}: \text{Minimize energy consumption } E \text{ and makespan } C_{max} = \max_{i \in tarea} (C_i)$$

Constrains: Problem is subject to constrains that guarantee that each task is assigned at least once [6], and that each machine only execute a task at a time [19].

Problem Instances. The problem instances are matrices ETC of size *txm* (t tasks and m machines) [17]. ETC model is characterized by the parameters consistency, machine heterogeneity and task heterogeneity [31]. Table 1 has a high variation along the column that is the high heterogeneity of tasks, and in Table 2, the large variation along a row represents high heterogeneity of machines.

Consistency is defined by the relation between a task and how it is executed in the machines according to heterogeneity of each one [8]. ETC matrix is consistent if a given machine m_j executes any task t_i faster than machine m_k, then machine m_j executes all tasks faster than machine m_k [31], as in Tables 1 and 2. If this occurs only partially, i.e., it has an inconsistent matrix that include a consistent sub-matrix, the matrix is considered semi-consistent. And if it does not have at least consistent submatrices, then it is an inconsistent instance.

Instances are labeled as *x_ttmm* [17], where *x* indicates the type of matrix consistency (c for consistent, s for semi-consistent, i for inconsistent), *tt* indicates the heterogeneity of tasks and *mm* indicates the heterogeneity of machines. For machines and tasks, "hi" and "lo" mean high and low heterogeneity respectively. With these heterogeneity and consistency features, twelve types of ETC instances are obtained.

Table 1. High heterogeneity of tasks and low heterogeneity of machines in ETC matrix of size 10 × 5

	m_1	m_2	m_3	m_4	m_5
t_1	11648,23	11803,25	13198,95	14208,43	15309,41
t_2	12826,31	13439,28	13326,27	15145,01	15323,84
t_3	10394,73	10543,99	10629,78	12025,45	14339,22
t_4	508,99	561,11	567,35	766,93	858,48
t_5	5084,65	5288,79	5872,92	6503,83	7001,72
t_6	1808,62	1869,03	1936,83	1987,72	2229,49
t_7	877,99	901,28	956,57	1039,97	1044,57
t_8	5331,69	5858,57	6379,28	6985,41	7339,16
t_9	25250,93	25747,22	25785,37	26322,56	26332,69
t_{10}	3905,32	4012,28	4016,58	4511,21	4521,13

Table 2. Low heterogeneity of tasks and high heterogeneity of machines in ETC matrix of size 10 × 5

	m_1	m_2	m_3	m_4	m_5
t_1	896,03	1033,62	3276,71	16061,46	25993,39
t_2	913,99	1573,82	2928,01	18939,34	27081,67
t_3	802,42	1220,04	2489,74	17588,49	25076,18
t_4	764,43	1389,37	2733,12	17863,56	27848,96
t_5	987,75	1524,07	3622,65	16750,89	24889,36
t_6	658,35	1379,73	2940,43	16916,91	23134,42
t_7	844,28	1437,73	2571,79	14899,55	25771,68
t_8	702,05	1504,82	2955,61	17555,64	25156,15
t_9	642,51	1053,21	3156,67	15995,97	26244,13
t_{10}	866,42	1589,23	2233,13	15738,73	26766,26

The most used instances are twelve ETC matrices of size 512 × 16 proposed by Braun et al. [8], one for each type of ETC instance.

2.3 Energy Model

Energy consumption of the objective function is determined through an energy model, which calculates the energy consumed by each processor in a time interval. Energy consumption by processors is defined as [2]:

$$E = P * \sum_{i=1}^{m} CT_i \tag{1}$$

Energy consumption depends of processor power P (watts) and how long it is operational CT (Completion Time). Operating system can self-regulate dynamically the supply voltage and clock frequency of the processor with techniques such as DVFS (Dynamic Voltage Frequency Scaling) [7, 13], to save energy and produce less heat. This is represented by discrete values of power, and processor power is adjusted to a minimum power when idle, and switches to the maximum power when processing a task.

In the work of Pinel and Bouvry [7], they proposed a more comprehensive energy model defined in Eq. 2, where BL is a constant power term, N is the number of machines powered on (a machine which is not used is considered powered off) [7], P_{high} y P_{low} is the CPU power consumption when operating at maximum y minimum voltage/frequency respectively.

$$E = BL * N * C_{max} + \sum_{i=1}^{m} (P_{high} * CT_i + P_{low} * (C_{max} - CT_i)) \tag{2}$$

3 Heuristic and Metaheuristic Methods for Job Scheduling in Grids

3.1 Heuristics of Job Scheduling

Some heuristic algorithms generate solutions from scratch by adding components to a partial solution, step by step, according to a transition rule until a solution is complete. The job scheduling problem present in clusters has been resolved by low complexity heuristics, which consume less time and memory to generate a schedule. A well-known heuristic is Min-Min, which begins with the set of all unmapped tasks, then the minimum expected completion time for each task of the set is establish, and the task with the lowest minimum completion time is selected and assigned to the corresponding machine, next the newly mapped task is removed from the set, and the process repeats until all tasks are mapped [8]. Max-Min works the same way that Min-Min, but according to the maximum expected completion time.

In a recent paper, Diaz et al. [17] compare Min-Min with *low complexity heuristics* Max-Max-Min, Avg-Max-Min and Min-Max-Min in Heterogeneous Computing Systems (HCS), and implemented Task Priority Diagram (TPD) algorithms. TPD defines a graph to set the precedence for each task based on the ETC value, using a Hasse diagram. Regarding makespan metric, *low complexity heuristics* were the best in *inconsistent* and *semi-consistent* scenarios, in consistent scenarios the TPD-based heuristics were better. Díaz et al. [15] compare Min-Min with the algorithms Min-Min-Min, Min-Mean-Min y Min-Max-Min, and were evaluated performance, energy efficiency and scalability in large-scale systems. Among this algorithms family were not presented significant differences in performance metrics (makespan and flowtime) and scalability, however, regarding the energy efficiency Min-Min was highlighted over the others.

Others specialized heuristics for job scheduling problems in distributed systems are Opportunistic Load Balancing (OLB), Minimum Execution Time (MET), Minimum Completion Time (MCT), Sufferage and High Standard Deviation First. In the extensive work of Braun et al. [8], where eleven heuristics are evaluated using the ETC model, a genetic algorithm obtained the lowest makespan, MCT heuristic outperformed to MET, and OLB got the worst makespan. OLB try to keep all machines as busy as possible, assigns each task to the next machine that is expected to be available, however, due to OLB does not consider expected task execution times, can result in a very long makespan [8]. MET try to assigns each task to the machine where is execute faster, i.e., the machine with the best expected execution time for that task, but because regardless of that machine's availability (current workload), this can cause a severe load imbalance across machines. MCT try to assigns each task to the machine with the minimum expected completion time for that task, in this manner seek to avoid the circumstances in which OLB and MET perform poorly [8]. But this causes some tasks to be assigned to machines that do not have the minimum execution time for them.

High Standard Deviation First (MaxStd) assigns first the task with the highest standard deviation of the expected execution time of the task, to the machine that has the minimum completion time, since the delay produced by their allocation will not

affect too much the total makespan. This standard deviation represents the amount variation in task execution time on different machines [2].

Sufferage is the difference between the best and the second-best minimum completion time of the task [2, 32]. Task with the highest sufferage is assigned to the task's second most favourable machine, because in other way would be the most delay.

3.2 Metaheuristics of Job Scheduling

HPC literature has more complex techniques known as metaheuristics, approaches that have been used to solve many optimization problems, and could be a basis to design efficient grid schedulers [20]. These find sub-optimal solutions of high quality, with less evaluations of solutions for combinatorial optimization problems, however, usually require long run times [20], much higher than run times of heuristics. The main metaheuristics that have been applied in job scheduling are shown in Table 3, along with their basic characteristics and related works. Some of these works follow the ETC model and most are about job scheduling in grid.

Some metaheuristics have random components, such as mutations in Evolutionary Algorithms, and additional information produced by itself, such as pheromone in Ant Colony Optimization, to guide and diversify the search for solutions. Even so, it cannot guarantee the finding of optimal solution, only can find approximate solutions. These methods depend much of quality and diversity of the initial solution, which is usually generated randomly to ensure diversity. Some methods are multi-boot, to explore other solutions to direct the search towards regions of the search space where the global optima is located, instead of getting stuck in a local optima. Metaheuristics can be based in local search and population.

Table 3. Basic characteristics of metaheuristics [12]

Metaheuristic	Characteristics	References
Simulated annealing	Acceptance criterion	30
	Cooling schedule	
Tabu search	Neighbor choice (tabu list)	29
	Aspiration criterion	
Evolutionary algorithms	Recombination	2, 8, 21, 22, 23, 24
	Mutation	
	Selection	
Ant colony optimization	Probabilistic construction Pheromone update	25, 26, 27
Particle swarm optimization	Population-based	28
	Social coefficient	

Metaheuristics Based in Local Search. A local search heuristic start from some initial solution and iteratively try to replace the current solution by a better solution in an appropriately defined neighborhood of the current solution [12]. Local Search

(LS) is performed until a stopping condition is met, such as a number of consecutive iterations without changing current solution or until the maximum execution time runs out. It only requires a few specifications as an evaluation function and an efficient method for exploring neighbourhood. This deterministic and memoryless method can find solutions quickly, but the final solution strongly depends on the initial solution to avoid getting stuck in the local optima and ensure convergence to suboptimal or optimal solutions.

Tabu Search (TS) in every iteration can accept higher cost solutions to explore other areas of the search space [35], taking into account a tabu list that prevents repeated moves. Xhafa et al. implemented this method under the ETC model [29].

Greedy Randomized Adaptive Search Procedure (GRASP) is a random iterative search method [31], which changes the current solution with a restricted candidate list (RCL) of the best options available, and ends when reach a stopping condition, e.g., achieve a given number of iterations.

Simulated Annealing (SA) is a stochastic search algorithm without any memory [30], inspired by the annealing process in metallurgy. In this process a material (such as steel) is heated to a specific temperature, the heat causes that atoms to increase their energy, and thus can easily move from their initial positions to explore the search space. Then it is gradually cooled until temperature environment, seeking to reach the global optima where material acquires desired physical properties (such as ductility, toughness, etc.). Algorithm starts from a random initial solution and a high probability (initial temperature) to allow any random move, which may be a worst quality solution than current solution, in order to escape the local minima and explore the search space. The probability to accept any movement gradually decrease (cooling) during the search, until become an iterative algorithm that accepts only current solution changes if there is an improvement. Cooling rule may change during the execution of the algorithm, in order to adjust the balance between diversification and intensification of search to converge to a solution [12].

Population-Based Metaheuristics. In population-based metaheuristics, the solution space is explored through a population of individuals. Main metaheuristics in this category are Evolutionary Algorithms, Ant Colony Optimization and Particle Swarm Optimization.

Evolutionary Algorithms (EA) are inspired by the evolution of living beings, so it uses selection and combination mechanisms. The most used of this family algorithms are genetic algorithms (GA), where from an initial population of chromosomes (solution), it seeks to find the most suitable (solution with the best cost in objective function) over the course of several generations, through crossover of chromosomes, random mutations of genes and selection of chromosomes that survive to produce the next generation. Genetic algorithms for the scheduling problem in grid has been quite used, e.g., by Braun et al. [8], Zomaya and Teh [21], Gao et al. [22] and Carretero et al. [23]. Pinel et al. [2] implemented a Genetic Algorithm in a conventional cluster, to which was added millicomputers to reduce power consumption. This algorithm is called PA-CGA (Parallel Asynchronous - Cellular Genetic Algorithm) and was proposed along with a heuristic called 2PH (Two Phase Heuristic), it consists of two phases, Min-Min followed by Local Search. Both algorithms were evaluated against Min-Min, achieving better

performance (makespan) with a low runtime. In the work of Nesmachnow et al. [37], proposed the Parallel Micro CHC (Cross generational elitist selection, Heterogeneous recombination and Cataclysmic mutation) algorithm and they obtained an excellent makespan for grid scheduling.

Other evolutionary computation algorithm is the Memetic Algorithm (MA), a hybrid algorithm that combines evolution ideas with local search, through memes (cultural information unit) similar to genes, common information of a population is transmitted to the next generation. Few works have implemented this algorithm for grid scheduling problem because it is a recent algorithm, such as Xhafa et al. [24] that proposes a Cellular MA (cMA) for scheduling under the ETC model.

In the literature several algorithms have been proposed following the Ant Colony Optimization (ACO) probabilistic method, to find approximate solutions to the combinatorial optimization problems as the tackled in our work. The first ACO algorithm was Ant System (AS) proposed by Marco Dorigo, and have been used to solve a similar problem called Job Shop Scheduling [26]. Recent versions gave better results, as the Max-Min Ant System (MMAS) [27]. An implementation of ACO for job scheduling in grid was conducted by Chang et al. [25].

Particle Swarm Optimization (PSO) is similar to ACO algorithm, which seeks to copy the swarming behavior of different living beings (bees, birds, fish, etc.). Abraham et al. proposed an approach for scheduling problem using a fuzzy PSO algorithm [28].

Also have implemented hybrid metaheuristics, mainly with Tabu Search. Other metaheuristic is executed first, e.g., a genetic algorithm that search a good quality solution, and then Tabu Search tries to improve it, exploring the neighborhood of that solution. In works that reported results with the Braun et al. instances [8], have been implemented hybrid metaheuristics MA + TS [34] and ACO + TS [36].

4 Comparison of Scheduling Algorithms

The instances used of Braun et al. benchmark are twelve of 512 jobs and 16 machines, which correspond to the twelve different types of ETC instances. The metrics analyzed in this work are makespan as the indicator of performance, and power consumption to establish energy efficiency according to the achieved performance. Energy efficiency was only determined for algorithms implemented by the Luxembourg University [15, 17], through the execution of algorithms using Braun et al. instances and the energy model (and parameters values) defined by Guzek et al. [40], because only the makespan is reported in most papers reviewed.

4.1 Comparative Analysis

The best makespan obtained for reported algorithms are compared in Table 4, which highlights the evolutionary algorithms Parallel CHC [33] and Parallel Micro CHC [37], the latter achieves the best makespan in all instances. Also it is highlighted Min-Min heuristic [1, 32], as it requires a very low running time to obtain good quality solutions, an issue in which the evolutionary metaheuristics are not very strong. The complete

results of all heuristics and metaheuristics are reported in the website http://forge.sc3.uis.edu.co/redmine/documents/1. There you can see that Min-Min is better than all heuristics as Sufferage and Max-Min, and it is known that Min-Min is also better than MET, MCT and OLB heuristics, according to the comparison graphs presented in the work of Braun et al. [8].

Makespan results of Min-Min and Max-Min reported in [32], agree with those obtained in the execution of algorithms provided by the Luxembourg University [15, 17]. Analyzed the makespan for each type of consistency, Min-Min and Sufferage heuristics have a long makespan in consistent and semi-consistent instances, Parallel CHC algorithm is the second best makespan in six of twelve instances, which mostly belong to the type of semi-consistent and inconsistent instances. In the remaining instances is overcome by Tabu Search [35], ACO + TS [36] and PA-CGA [38]. Although the hybrid metaheuristics (ACO + TS and cMA + TS) are not the best in this case, they are a good alternative to be further investigated.

In algorithms that we have the necessary information to assess the multi-objective function, which aims to minimize energy consumption and makespan simultaneously, a score function SF is used [7, 17]. It represents the energy efficiency, also known as fitness function [20], to set the importance of the two objectives with a weight parameter α as follows:

$$SF = \alpha * C_{max} + (1 - \alpha) * E \tag{3}$$

Therefore, the aim will be to minimize the score function. If the priority of both objectives are the same, we can set α at 0.5 to have a balanced weight or importance of makespan and energy. Moreover, it is required to normalize the values of each metric for appropriate calculation of the score (because metrics have different measure units), so the value of makespan and energy obtained by each algorithm is divided by the maximum value of all algorithms [15]. Normalized values are in the range [0,1], where 1 is the worst performance value. The score obtained from the executed algorithms is compared in Fig. 1. Min-Min algorithm is better in all instances with a balanced α at 0.5, and as well with α between 0.1 and 0.9 approximately. Also is outstanding the energy efficiency of Min-Mean-Min and Min-Max-Min. In contrast, Max-Min has one of the worst performance, especially in inconsistent instances where it has the highest energy consumption and makespan.

4.2 Analysis of the Highlighted Algorithms

The highlighted algorithms according to the makespan are Parallel Micro CHC and Parallel CHC algorithms. These evolutionary algorithms have in common its basics characteristics (selection, recombination and mutation), and achieve to find the best solutions for grid scheduling problem, because they have a good balance between random and guided search. Starting from a random initial solution (or obtained with fast heuristics as Min-Min, MCT and Sufferage), explore the search space guided by the information contained in a population of chromosomes. For this is defined a rule to select the fittest individuals, which are recombine to generate new individuals

Table 4. Best makespan of algorithms

Instance 512 × 16	Min-Min [32]	Sufferage [32]	TS [35]	cMA [24]	GA [33]	Parallel GA [33]	PA-CGA [38]	CHC [33]	Parallel CHC [33]	Parallel Micro-CHC [37]	MA + TS [34]	ACO + TS [36]
u_c_hihi.0	8.460.674	10.908.698	7.448.641	7.700.930	7.659.879	7.577.922	7.437.591	7.599.288	7.461.819	**7.381.570**	7.530.020	7.497.201
u_c_hilo.0	161.805	167.483	153.263	155.335	155.092	154.915	154.393	154.947	153.792	**153.105**	153.917	154.235
u_c_lohi.0	275.837	349.746	241.673	251.360	250.512	248.772	242.062	251.194	241.513	**239.260**	245.289	244.097
u_c_lolo.0	5.441	5.650	5.155	5.218	5.239	5.208	5.248	5.226	5.178	**5.148**	5.174	5.178
u_i_hihi.0	3.513.919	3.391.758	2.957.854	3.186.665	3.019.844	2.990.518	3.011.581	3.015.049	2.952.493	**2.938.381**	3.058.475	2.947.754
u_i_hilo.0	80.756	78.828	73.693	75.857	74.143	74.030	74.477	74.241	73.640	**73.378**	75.109	73.776
u_i_lohi.0	120.518	125.689	103.866	110.621	104.688	103.516	104.490	104.546	102.123	**102.051**	105.809	102.446
u_i_lolo.0	2.786	2.674	2.552	2.624	2.577	2.575	2.603	2.577	2.549	**2.541**	2.597	2.554
u_s_hihi.0	5.160.343	5.574.358	4.168.796	4.424.541	4.332.248	4.262.338	4.229.018	4.299.146	4.198.780	**4.103.500**	4.321.015	4.162.548
u_s_hilo.0	104.375	103.401	96.181	98.284	97.630	97.506	97.425	97.888	96.623	**95.787**	97.177	96.762
u_s_lohi.0	140.285	153.094	123.407	130.015	126.438	125.717	125.579	126.238	123.237	**122.083**	127.633	123.922
u_s_lolo.0	3.807	3.728	3.451	3.522	3.510	3.480	3.526	3.492	3.450	**3.434**	3.484	3.455
Average	1.502.545	1.738.759	1.281.544	1.345.414	1.319.317	1.303.875	1.290.666	1.311.153	1.284.600	**1.268.353**	1.310.475	1.284.494

Note: Bold values are the best results

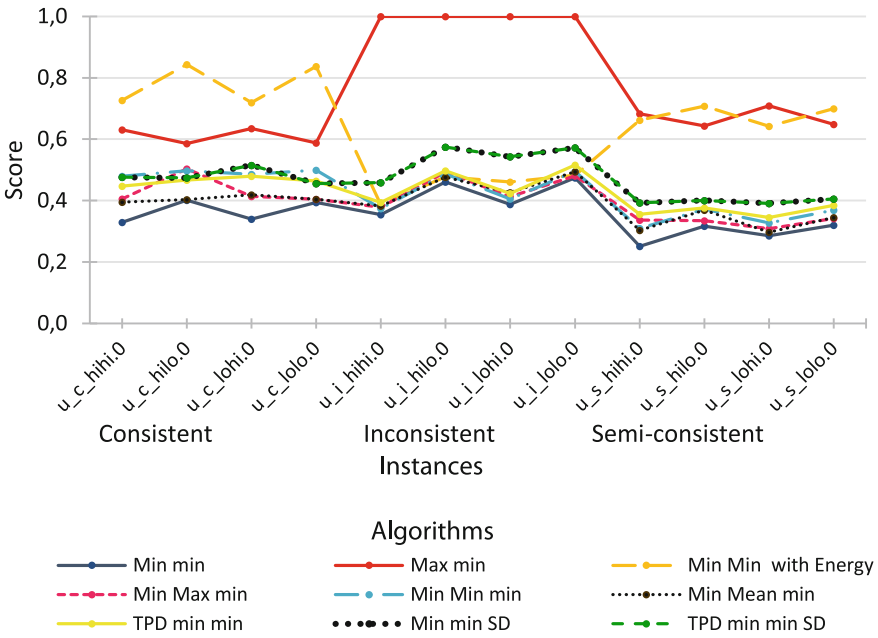

Fig. 1. Energy efficiency of algorithms

(offspring), and with the passing of generations (iterations) it allows to reach a high quality solution. Mutations that occur randomly in traditional evolutionary algorithms, to overcome local optima and diversify the search, in these algorithms are replaced by a mating restriction between very similar individuals and a reset process when the search tends to quickly converge to local optima.

Both algorithms differ mainly in that Parallel Micro CHC includes additional concepts of Micro Genetic Algorithm (μ-GA) [41], to avoid getting stuck in by the lack of diversity in the solutions when small populations are used, through the maintenance of an elite population used to reset the main population each specific number of generations.

Between the heuristics, Min-Min got a good balance between energy consumption and makespan, due to it always first assigns the task to the machine with the overall minimum completion time [8], therefore, the system has more available machines to execute tasks in the best corresponding machine, i.e., the machine with the lowest ETC for the task. Min-Min likely can assign more tasks to their best ETC than Max-Min, which first assigns the tasks to the machine with the maximum completion time. Min-Min heuristic assigns the first task t_i to the machine that finishes it earlier and executes it faster, and for every task assigned after t_i, Min-Min changes the machine availability status by the least possible amount for every assignment. The expectation is that a smaller makespan can be obtained if more tasks are assigned to the machines that complete them the earliest and also execute them the fastest [8].

5 Conclusions and Future Works

This article presented an overview of the most important heuristic and metaheuristic methods to solve the job scheduling problem in grid computing. The algorithms have been compared and analyzed in terms of job scheduling under the ETC model (with the most common instances). In these terms, the evolutionary algorithm Parallel Micro CHC is the best method identified according to the makespan, and full results reported suggest that the evolutionary algorithms are well suited to face the complexity of scheduling problem. The main heuristics are also compared according to the energy efficiency, where the Min-Min algorithm stands out over the other heuristics executed.

With this review article, new researchers can determine the heuristics most prominent nowadays, to implement their diverse search strategies in related combinatorial optimization problems. The main lines for future work include design an evolutionary algorithm of low-complexity to get an appropriated execution time on a low-power computational infrastructure, and minimize both the makespan and energy consumption. The purpose is get a better balance in all types of instances and improve the energy efficiency in HPC resources, so we are working on an ARM-based processors cluster and we will propose an energy model based in experimental data obtained using this platform.

Acknowledgments. The authors thank to the University of Luxembourg for providing us with algorithms to test their performance with instances of Braun et al. benchmark.

References

1. Pinel, F., Pecero, J.E., Khan, S.U., Bouvry, P.: Energy-efficient scheduling on milliclusters with performance constraints. In: Proceedings of the 2011 IEEE/ACM International Conference on Green Computing and Communications, pp. 44–49 (2011)
2. 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. Cluster Comput. **16** (3), 421–433 (2013)
3. Izakian, H., Abraham, A., Snasel, V.: Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. In: International Joint Conference on Computational Sciences and Optimization, vol. 1, pp. 8–12 (2009)
4. He, X., Sun, X., Von Laszewski, G.: QoS guided min-min heuristic for grid task scheduling. J. Comput. Sci. Technol. **18**(4), 442–451 (2003)
5. Iqbal, S., Gupta, R., Lang, Y.: Job scheduling in HPC clusters. Power Solutions, pp. 133–135 (2005)
6. Dutot, P.F., Eyraud, L., Mounié, G., Trystram, D.: Bi-criteria algorithm for scheduling jobs on cluster platforms. In: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, pp. 125–132 (2004)
7. Pinel, F., Bouvry, P.: A model for energy-efficient task mapping on milliclusters. In: Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, pp. 1–32 (2011)

8. Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M., Reuther, A.I., Freund, R. F.: 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)
9. Diaz, C.O., Guzek, M., Pecero, J.E., Danoy, G., Bouvry, P., Khan, S.U.: Energy-aware fast scheduling heuristics in heterogeneous computing systems. In: 2011 International Conference on High Performance Computing and Simulation (HPCS), pp. 478–484 (2011)
10. Leung, J.Y. (ed.): Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Boca Raton (2004)
11. Ali, S., Braun, T.D., Siegel, H.J., Maciejewski, A.A., Beck, N., Bölöni, L., Yao, B.: Characterizing resource allocation heuristics for heterogeneous computing systems. Adv. Comput. **63**, 91–128 (2005)
12. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. (CSUR) **35**(3), 268–308 (2003)
13. Valentini, G.L., Lassonde, W., Khan, S.U., Min-Allah, N., Madani, S.A., Li, J., Bouvry, P.: An overview of energy efficiency techniques in cluster computing systems. Cluster Comput. **16**(1), 3–15 (2013)
14. Hussain, H., Malik, S.U.R., Hameed, A., Khan, S.U., Bickler, G., Min-Allah, N., Rayes, A.: A survey on resource allocation in high performance distributed computing systems. Parallel Comput. **39**(11), 709–736 (2013)
15. Diaz, C.O., Guzek, M., Pecero, J.E., Bouvry, P., Khan, S.U.: Scalable and energy-efficient scheduling techniques for large-scale systems. In: 11th International Conference on Computer and Information Technology (CIT), pp. 641–647 (2011)
16. Barrondo, A., Tchernykh, A., Schaeffer, E., Pecero, J.: Energy efficiency of knowledge-free scheduling in peer-to-peer desktop Grids. In: 2012 International Conference on High Performance Computing and Simulation (HPCS), pp. 105–111 (2012)
17. Diaz, C.O., Pecero, J.E., Bouvry, P.: Scalable, low complexity, and fast greedy scheduling heuristics for highly heterogeneous distributed computing systems. J. Supercomputing **67**(3), 837–853 (2014)
18. Dong, F., Akl, S.G.: Scheduling algorithms for grid computing: state of the art and open problems. School of Computing, Queen's University, Kingston, Ontario (2006)
19. Lindberg, P., Leingang, J., Lysaker, D., Bilal, K., Khan, S.U., Bouvry, P., Li, J.: Comparison and analysis of greedy energy-efficient scheduling algorithms for computational grids. In: Energy-Efficient Distributed Computing Systems, pp. 189–214 (2011)
20. Xhafa, F., Abraham, A.: Computational models and heuristic methods for grid scheduling problems. Future Gener. Comput. Syst. **26**(4), 608–621 (2010)
21. Zomaya, A.Y., Teh, Y.H.: Observations on using genetic algorithms for dynamic load-balancing. IEEE Trans. Parallel Distrib. Syst. **12**(9), 899–911 (2001)
22. Gao, Y., Rong, H., Huang, J.Z.: Adaptive grid job scheduling with genetic algorithms. Future Gener. Comput. Syst. **21**(1), 151–161 (2005)
23. Carretero, J., Xhafa, F., Abraham, A.: Genetic algorithm based schedulers for grid computing systems. Int. J. Innovative Comput. Inf. Control **3**(6), 1–19 (2007)
24. Xhafa, F., Alba, E., Dorronsoro, B., Duran, B., Abraham, A.: Efficient batch job scheduling in grids using cellular memetic algorithms. In: Metaheuristics for Scheduling in Distributed Computing Environments, pp. 273–299 (2008)
25. Chang, R.S., Chang, J.S., Lin, P.S.: An ant algorithm for balanced job scheduling in grids. Future Gener. Comput. Syst. **25**(1), 20–27 (2009)
26. Colorni, A., Dorigo, M., Maniezzo, V., Trubian, M.: Ant system for job-shop scheduling. Belg. J. Oper. Res. Stat. Comput. Sci. **34**(1), 39–53 (1994)

27. Stützle, T., Hoos, H.H.: MAX–MIN ant system. Future Gener. Comput. Syst. **16**(8), 889–914 (2000)
28. Liu, H., Abraham, A., Hassanien, A.E.: Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. Future Gener. Comput. Syst. **26**(8), 1336–1343 (2010)
29. Xhafa, F., Carretero, J., Dorronsoro, B., Alba, E.: A tabu search algorithm for scheduling independent jobs in computational grids. Comput. Inform. **28**, 237–250 (2009)
30. Kirkpatrick, S., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
31. Ali, S., Siegel, H.J., Maheswaran, M., Hensgen, D., Ali, S.: Representing task and machine heterogeneities for heterogeneous computing systems. Tamkang J. Sci. Eng. **3**(3), 195–208 (2000)
32. Xhafa, F., Barolli, L., Durresi, A.: Batch mode scheduling in grid systems. Int. J. Web Grid Serv. **3**(1), 19–37 (2007)
33. Nesmachnow, S., Cancela, H., Alba, E.: Heterogeneous computing scheduling with evolutionary algorithms. Soft. Comput. **15**(4), 685–701 (2010)
34. Xhafa, F.: A hybrid evolutionary heuristic for job scheduling on computational grids. In: Hybrid Evolutionary Algorithms, pp. 269–311 (2007)
35. Xhafa, F., Carretero, J., Alba, E., Dorronsoro, B.: Design and evaluation of tabu search method for job scheduling in distributed environments. In: Proceedings of the 22th International Parallel and Distributed Processing Symposium, pp. 1–8 (2008)
36. Ritchie, G., Levine, J.: A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. In: Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group, pp. 178–183 (2004)
37. Nesmachnow, S., Cancela, H., Alba, E.: A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. Appl. Soft Comput. **12**(2), 626–639 (2012)
38. Pinel, F., Dorronsoro, B., Bouvry, P.: A new parallel asynchronous cellular genetic algorithm for scheduling in grids. In: 2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and PhD Forum, pp. 1–8 (2010)
39. Bardsiri, A.K., Hashemi, S.M.: A comparative study on seven static mapping heuristics for grid scheduling problem. Int. J. Softw. Eng. Appl. **6**(4), 247–256 (2012)
40. Guzek, M., Pecero, J.E., Dorronsoro, B., Bouvry, P.: Multi-objective evolutionary algorithms for energy-aware scheduling on distributed computing systems. Appl. Soft Comput. **24**, 432–446 (2014)
41. Coello Coello, C.A., Toscano Pulido, G.: A micro-genetic algorithm for multiobjective optimization. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) EMO 2001. LNCS, vol. 1993, pp. 126–140. Springer, Heidelberg (2001)