A MULTITHREADING LOCAL SEARCH FOR MULTIOBJECTIVE ENERGY-AWARE SCHEDULING IN HETEROGENEOUS COMPUTING SYSTEMS

Santiago Iturriaga, Sergio Nesmachnow
Universidad de la República
Montevideo, Uruguay
Email: {siturria,sergion}@fing.edu.uy

Bernabé Dorronsoro
Interdisciplinary Centre for Security, Reliability, and Trust
Luxembourg
Email: bernabe.dorronsoro@uni.lu

KEYWORDS

multithreading, local search, scheduling, heterogeneous computing

ABSTRACT

This article introduces an efficient multithreading local search algorithm for solving the multiobjective scheduling problem in heterogeneous computing systems considering the makespan and energy consumption objectives. The proposed method follows a fully multiobjective approach using a Pareto-based dominance search executed in parallel. The experimental analysis demonstrates that the new multithreading algorithm outperforms a set of deterministic heuristics based on Min-Min. The new method is able to achieve significant improvements in both objectives in reduced execution times for a broad set of testbed instances.

INTRODUCTION

Nowadays, distributed heterogeneous computing (HC) platforms gather hundreds or thousands of computing resources from different organizations located worldwide. In the last decade, *grid* infrastructures emerged as a useful choice to provide the computing power needed to tackle complex problems in many application domains (e.g. scientific, industrial, commercial, etc.) (Foster and Kesselman, 1998).

The efficient management of the large number of available resources in HC and grid infrastructures poses several challenges, and this can be a problem as complex as the applications to be executed/solved over the infrastructure. The efficient allocation of tasks to be executed in the distributed resources of an HC grid infrastructure is a key problem to solve in order to take full advantage of the computing power of the grid. This kind of task-worker allocation problems, called *scheduling* problems, have been thoroughly studied in the operational research field. However, most of the classic approaches tackle only homogeneous environments (El-Rewini et al., 1994; Leung et al., 2004). The distributed heterogeneous computing systems present a new heterogenity characteristic, and in this scenario the classic aproaches are clearly outperformed by many newly defined methods (Dorronsoro et al., 2010; Nesmachnow et al., 2012).

The goal of the scheduling problem is to assign tasks to the computing resources by satisfying some efficiency criteria, usually related to the total execution time for a bunch of tasks (makespan), but frequently also considering other metrics. Traditional scheduling problems are NP-hard, thus exact methods are not useful in practice to solve large instances. Deterministic heuristics are able to compute accurate results, but in general they do not scale nicely to large dimension instances. Thus, non-deterministic heuristics and metaheuristics are used to tackle scheduling problems, and has been shown that these methods are able to produce efficient schedules in reasonable timespans for both small and large dimension instances (Nesmachnow et al., 2010; Nesmachnow and Iturriaga, 2011).

Energy consumption has become a great challenge in distributed high performance computing, and researchers have focused on developing energy-aware scheduling algorithms for distributed HC infrastructures (Lee and Zomaya, 2009). In this work, we design and evaluate a highly efficient multiobjective local search (LS) metaheuristic to find accurate tradeoff solutions to the multiobjective scheduling problem of minimizing the makespan and the energy consumption in HC and grid systems (ME-HCSP). The proposed method follows a fully multiobjective approach, since it does not optimize an aggregated function of the problem objectives, but uses a Pareto-based dominance analysis instead.

The main contributions of this manuscript are: i) to introduce a fast local search method for energy-aware scheduling in HC grid systems, and ii) to compute accurate schedules in reduced execution times following a truly multiobjective approach. Using only 10 seconds of execution time, the new algorithm is able to outperform the results provided by the compared deterministic heuristics, obtaining improvements of up to **19%** for the makespan objective, and up to **9%** for the energy consumption objective.

The manuscript is organized as follows. Next section introduces the energy-aware scheduling problem in HC systems and a review of related work. After that, the new multiobjective LS proposed in this work is presented. Later, the experimental evaluation of the proposed method over a large set of problem instances is described. Finally, the conclusions of the research are presented and the main lines for future work are formulated.

Proceedings 26th European Conference on Modelling and Simulation ©ECMS Klaus G. Troitzsch, Michael Möhring, Ulf Lotzmann (Editors)
ISBN: 978-0-9564944-4-3 / ISBN: 978-0-9564944-5-0 (CD)

ENERGY-AWARE SCHEDULING IN HC SYSTEMS

Problem Formulation

An HC system is composed of computers (*machines*), with different computing power and energy consumption, depending on the hardware features. In the independent batch scheduling problem, a set of non-dependent tasks with variable computing demands must be scheduled to execute on the system. A task cannot be divided, nor interrupted after it is assigned to a machine (*non-preemptive* scheduling). The execution times of any task and the energy to perform it vary from one machine to another. Thus, there will be competition for using those machines able to execute tasks in short time and with low energy consumption.

In scheduling, the most usual metric to minimize is the *makespan*, defined as the time spent since the first task begins execution to the moment when the last task is completed (Leung et al., 2004). The mathematical model for the ME-HCSP considers the following elements:

- An HC system composed of a set of heterogeneous machines $P = \{m_1, \ldots, m_M\}$; and 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 $EC_{IDLE}(m_j)$ is the energy that machine m_j consumes in idle state.

The ME-HCSP proposes to find a schedule $f : T^N \rightarrow P^M$ that simultaneously minimizes the *makespan* (Eq. 1), and the *energy consumption* (Eq. 2), which accounts for both the energy required for the tasks' execution and the energy that each machine spends in idle state. The energy for executing a given task depends on the execution time, but these two objectives are usually in conflict, since fast machines generally consume more energy than the slower ones.

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

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

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

The independent tasks model also arises when users submit their tasks to execute in a computing service, specially in grid and volunteer-based infrastructures—WLCG, Teragrid, BOINC, Xgrid (Berman et al., 2003)—, where domain decomposition applications are often submitted for execution. Thus, the ME-HCSP faced in this work is relevant in realistic distributed HC and grid environments.

Related Work

The main trend in existing energy-aware schedulers is to apply energy management methods in the computing elements, such as dynamic voltage scaling (DVS), dynamic power management, and slack sharing/reclamation (Kim et al., 2007; Khan and Ahmad, 2009).

Kim et al. (2008) introduced several online/batch dynamic heuristics for scheduling tasks with priorities and deadlines in an ad-hoc grid with limited battery capacity, using DVS for power management. The batch dynamic heuristics based on MinMin (Luo et al., 2007) performed the best, but they required significantly more time. Li et al. (2009) also proposed an energy-aware scheduler based on MinMin to reduce energy consumption using dynamic power management.

Khan and Ahmad (2009) developed an energy-aware grid scheduler based on the concept of Nash Bargaining Solution from cooperative game theory, for DVS-enabled machines. Mezmaz et al. (2011) applied a cooperative parallel biobjective hybrid genetic algorithm (GA), improved with the energy-aware heuristics by Lee and Zomaya (2011). Pecero et al. (2011) applied a biobjective Greedy Randomized Adaptive Search Procedure scheduler, which builds a feasible solution using a greedy evaluation function, and uses a biobjective LS using DVS to improve the solution quality and generate a set of Pareto solutions.

A two-phase energy-aware heuristic for grid scheduling was proposed by Pinel et al. (2011), by first applying MinMin to find good makespan, and a LS in the second phase. Schedules with similar quality to those computed by a cellular GA are found in less time, while significantly improving the MinMin schedules. Kessaci et al. (2011) propose two multiobjective parallel GAs enhanced with energy-aware scheduling heuristics for tasks with dependencies, combining makespan and energy consumption in a unique function and using explicit DVS for energy management.

Kolodziej et al. (2011) developed a GA framework for energy-aware schedulers using DVS for energy reduction, a bag-of-tasks model, and a biobjective scheduling problem minimizing makespan and energy consumption. Recently, Lindberg et al. (2012) introduced a set of eight heuristics, including six list scheduling algorithms and two GAs, for the energy-aware grid scheduling problem subject to deadline constraint and tasks' memory requirements.

The approach in our article does not apply DVS or other energy management methods. Instead, we propose to explicitly compute the energy consumption considering the Max-Min mode, which accounts for hardware-embedded energy saving features (e.g. SpeedStep technology by Intel, or Optimized Power Management by AMD): a working machine is assumed to operate at peak performance, consuming the maximum energy, and when a machine is idle, the embedded energy saving technology keeps the energy consumption at its minimum specified value.

A MULTITHREADING ENERGY-AWARE SCHEDULER FOR HC SYSTEMS

This section describes the multithreading LS algorithm to solve the energy-aware scheduling problem in HC systems.

ME-MLS Design

ME-MLS is a population-based LS algorithm, which maintains a population of non-dominated schedules in order to avoid biasing the search toward one of the objectives of the energy-aware scheduling problem. The algorithm uses a pool of threads in which each thread is a peer, and no thread performs a master role. Algorithm 1 presents the pseudo-code of each thread in ME-MLS.

Algorithm 1 ME-MLS algorithm for each thread.

```
1:  thread_idx ← current thread index
2:  initialize_individual(population, thread_idx)
3:  initialization_barrier()
4:  while not stop_criteria do
5:      lock_population()
6:      s ← draw_random_schedule(population)
7:      s' ← clone(s)
8:      unlock_population()
9:      search_strategy ← random_strategy()
10:     max_iterations ← random(1, THREAD_ITER)
11:     while not search_ready do
12:         for i = 0 → max_iterations do
13:             local_search(search_strategy, s')
14:         end for
15:         trylock_population()
16:         if locked then
17:             check_Pareto_dominance(population, s')
18:             search_ready ← true
19:             unlock_population()
20:         else
21:             rework ← THREAD_ITER / REWORK_FACTOR
22:             max_iterations ← random(1, rework)
23:             search_ready ← false
24:         end if
25:     end while
26: end while
```

Each thread starts by initializing a schedule in the population using a randomized version of the Minimum Completion Time (MCT) heuristic. After that, each thread loops over the schedules in the population searching to improve them. On each loop step, each thread randomly selects a schedule s from the population to perform a LS. To perform the search, the thread clones the selected solution $s' = s$ and then applies a random number of iterations of the LS upon the clone s'. If a given thread finds a non-dominated schedule, it is inserted into the population; otherwise it is discarded. Then, the thread loops and once again randomly selects a schedule to improve from the population of non-dominated schedules.

The population of non-dominated schedules is limited in size, so a replacement policy is applied when the maximum size is reached. Each new non-dominated schedule is always inserted in the population. When the population is full, a schedule currently in the population is selected to be replaced based on a distance function, helping to prevent a stagnation situation. Those schedules that compute the minimum value in the population in at least one of the objectives are never replaced. Figure 1 shows the main activities performed during the execution of the ME-MLS algorithm.

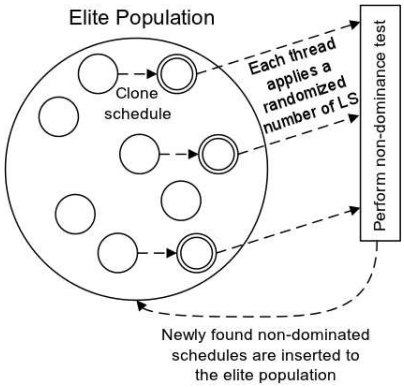


Figure 1: Diagram of the ME-MLS algorithm.

Local search

The energy-aware LS performed by each thread is based on a randomized version of the Problem Aware Local Search proposed by Alba and Luque (2007). Algorithm 2 presents the pseudo-code of the LS performed by each thread.

Algorithm 2 Energy-aware local search.

```
1:  strategy ← current search strategy
2:  s' ← schedule to improve
3:  mach_a ← random_machine(strategy, s')
4:  tasks_x ← random_task_set(mach_a)
5:  mach_b ← random_machine(strategy, s') ≠ mach_a
6:  tasks_y ← random_task_set(mach_b)
7:  best_operation ← none
8:  for all x ∈ tasks_x do
9:      operator ← random(move or swap)
10:     if operator = swap then
11:         for all y ∈ tasks_y do
12:             delta_{x,y} ← improvement(strategy, s', swap_{x,y})
13:             delta_best ← improvement(strategy, s', best_operation)
14:             if delta_{x,y} > delta_best then
15:                 best_operation ← swap_{x,y}
16:             end if
17:         end for
18:     else if operator = move then
19:         dst_set ← random_machine_set() ∪{mach_b}
20:         for all dst ∈ dst_set do
21:             delta_{x,dst} ← improvement(strategy, s', move_{x,dst})
22:             delta_best ← improvement(strategy, s', best_operation)
23:             if delta_{x,dst} > delta_best then
24:                 best_operation ← move_{x,dst}
25:             end if
26:         end for
27:     end if
28: end for
29: if best_operation ≠ none then
30:     apply_operation(best_operation)
31: end if
```

When applied to a schedule s, the energy-aware LS selects one of the following optimization strategies: (1) makespan optimization, (2) energy optimization, or (3) random optimization. Then, the method iterates a randomized number of times trying to improve the schedule s. In each iteration, the method selects a pair of machines m_A and m_B to perform the search. When the makespan optimization strategy is selected, it selects with high probability the makespan defining machine as m_A and the machine with lower computing time as machine m_B. The energy optimization strategy selects with high probability the most energy consuming machine as m_A and the less energy consuming machine as m_B. Finally, the random optimization strategy selects two random machines as m_A and m_B. After that, the method selects a random set of tasks $tasks_X$ from the ones assigned to $machine_A$. For each task $t_X \in tasks_X$, the method randomly selects an operation to perform: (1) a swap operation, or (2) a move operation. If the swap operation is selected for task t_X, the method selects a random set of tasks $tasks_Y$ assigned to $machine_B$ and computes the task t_X swap with each task $t_Y \in tasks_Y$. The method chooses the swap which improves the most the schedule objectives. If the energy and makespan improvements of the computed swap are in conflict (i.e. the swap improves one metric but degrades the other), the method selects the best swap prioritizing one metric according to the selected search strategy. When no swap is found to improve neither of the schedule objectives, no swap is applied.

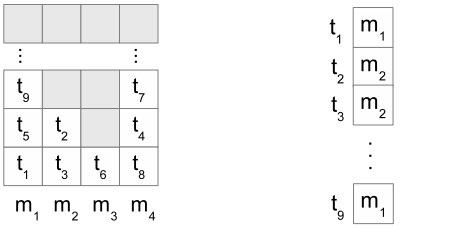
If the move operation is selected for task t_X, the method selects a set of machines $machines_Y = \{random\ set\ of\ machines\} \cup \{m_B\}$, and computes the variation in the schedule objectives when moving the task t_X to each machine $m_Y \in machines_Y$. The method chooses the move operation which improves the most the schedule objectives prioritizing the currently selected search strategy. Again, if no move is found to improve neither of the schedule objectives, then no move is applied.

Implementation Details

This subsection presents the implementation details of the ME-MLS algorithm proposed in this work.

Implementation language and libraries. The ME-MLS algorithm is implemented in GNU C++ 4.6, but certain C++ constructs where avoided in order to minimize the code execution overhead (e.g.: classes, interfaces, polymorphism, etc.). The multithreading support is provided by the GNU POSIX thread library 2.13. The use of higher level libraries, like OpenMP, were avoided in order to gain fine grain control over the synchronization mechanisms. Finally, the Mersenne Twister (MT) method (Matsumoto and Nishimura, 1998) is used for generating pseudorandom numbers. For this work, the original MT implementation was modified to be thread-safe, to allow the multithreading generation of random numbers.

Problem encoding. To store a schedule, a multi-structure is maintained in memory comprising both a machine-based and a task-based encoding (Nesmachnow et al., 2010) (see Figure 2). This in-memory multi-structure allows to: i) access the tasks assigned to a given machine in $O(1)$ execution time via the machine-based encoding; and ii) given a certain task, locate the machine to which is assigned also in $O(1)$ execution time via the task-based encoding.



(a) Machine-oriented encoding. (b) Task-oriented encoding.

Figure 2: Problem encodings.

Population initialization. A randomized version of the MCT heuristic is used to initialize the population. The randomized MCT considers the tasks sorted in a random order, thus tasks are scheduled also in a random order, providing some diversity in the initial population.

EXPERIMENTAL ANALYSIS

This section introduces the set of ME-HCSP instances and the hardware platform used in the experimental analysis. After that, the set of Min-Min based heuristics for energy-aware scheduling used as a reference baseline to compare the results computed by the ME-MLS algorithm are introduced. Finally, the experimental results are presented and analyzed, along with a scalability study that evaluates the speedup of the proposed method when using different numbers of threads.

ME-HCSP instances

To evaluate the proposed ME-MLS algorithm a set of **792** ME-HCSP instances were generated. Each instance describes the machines *scenario* and the set of tasks *workloads*. The scenarios were generated using realistic data, gathering for each machine the computational power, the energy consumption in idle state, and the energy consumption in processing state. The workloads were generated following the ETC methodology proposed by Ali et al. (2000).

Three instance dimensions were evaluated (*number of tasks* $\times$ *number of machines*): 512$\times$16, 1024$\times$32, and 2048$\times$64. A number of 11 different scenarios were generated for each dimension, and 24 workloads were generated for each problem dimension, 12 following the parametrization by Ali et al. (2000) and 12 following the parametrization by Braun et al. (2001).

Execution hardware platform
The experimental analysis of the ME-MLS algoritm was performed on a 24-Core AMD Opteron Processor 6172, 2.1GHz, 24 GB RAM, running 64-bits CentOS 5.1 Linux.

Min-Min based heuristics
In order to compare the ME-MLS results, we proposed a set of heuristics based on the Min-Min (Luo et al., 2007) heuristic which is considered to outperform other deterministic heuristics for minimizing the makespan objective (Izakian et al., 2009).

Min-Min uses a two-phase optimization strategy, thus we define four versions of the Min-Min heuristic alternating the minimization objective.The first version minimizes the completion time in both phases; the second version minimizes the energy objective in both phases; in the remaining versions the minimization objectives are alternated to be in the first phase or the second phase. When the completion time is minimized we use the *Min* notation, and when the energy metric is minimized we use the *MIN* notation is used. Thus the four Min-Min versions are: *Min-Min*, *MIN-MIN*, *Min-MIN*, and *MIN-Min*.

Parameter settings
The objective of this work is to *efficiently* solve the ME-HCSP, thus a fixed 10 seconds execution time stopping criterion is used for the ME-MLS algorithm. This time stopping criterion is significantly lower than the execution time of the algorithms in the related literature. A number of 30 independent ME-MLS executions were performed on each instance, considering the stochastic nature of the proposed algorithm. Each execution was performed using 24 threads, the maximum number of cores available.

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

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

Results and discussion
Table 1 presents the average improvements obtained when comparing the ME-MLS algorithm with the Min-Min based heuristic that performs the best for that instance and objective. Figure 3 summarizes the average improvement by dimension compared to each Min-Min based heuristic.

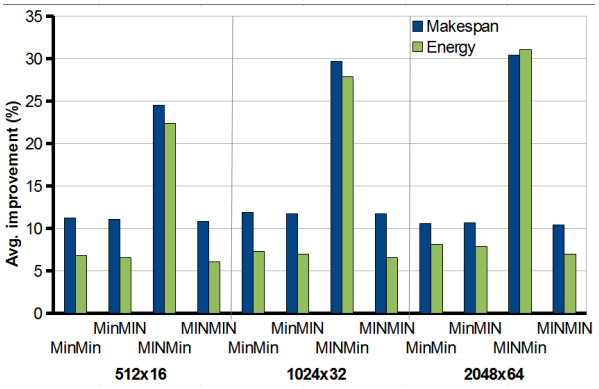


Figure 3: Average ME-MLS improvement over the Min-Min based heuristics.

Table 1 demonstrates that ME-MLS outperforms the best Min-Min based heuristic on the makespan metric with improvements ranging from 3.5% to **19.8%** and on the energy metric with improvements ranging from 2.6% to **9.4%**. Table 1 also shows that ME-MLS produces the best improvements when solving the inconsistent instances, and the second best improvements are produced when solving the semiconsistent instances.

Regarding the energy metric, ME-MLS achieved an acceptable steady improvement ranging from 3.4% to **8.0%** when grouping by type and herogeneity, and ranging from 5.8% to **6.8%** when grouping by dimension. On the other hand, the makespan metric is most sensible to the heterogeneity and type variation, and shows a considerable gap ranging from 4.7% to 16.6% when considering different types and heterogeneities.

Another interesting observation is that the ME-MLS algorithm improvements actually increase with the problem dimension. This performance increase is manly due to the improvements obtained when solving the inconsistent type instances; the improvements when solving this type of instances constantly increase with the dimension increase.

Regarding the computational efficiency of the ME-MLS algorithm, a speedup study was performed. The speedup evaluation was performed using 1024×32 instances, and performing 30 independent executions with a stopping criterion of 6 million iterations. The analysis shows the ME-MLS algorithm has a near linear speedup, achieving a speedup of 19.7 when using 24 threads. Figure 4 shows the results of the speedup analysis.

CONCLUSIONS

This work presented ME-MLS, a multiobjective multithread LS algorithm with strong focus on producing accurate results in short execution times (i.e. less than 10 seconds) for the energy-aware scheduling problem in HC systems. The ME-MLS algorithm uses a population of non-dominated schedules and a Pareto-based search following a fully multiobjective approach.

Table 1: Average ME-MLS improvement over the best Min-Min based heuristic.

type	heterogeneity	dimension						average	
		512×16		1024×32		2048×64			
		makespan	*energy*	*makespan*	*energy*	*makespan*	*energy*	*makespan*	*energy*
consistent	high high	9.2%	5.3%	6.6%	6.3%	5.6%	7.0%	7.1%	6.2%
	high low	6.5%	3.7%	7.1%	5.9%	5.8%	7.4%	6.5%	5.7%
	low high	8.1%	4.7%	7.4%	6.0%	6.0%	8.2%	7.2%	6.3%
	low low	4.5%	5.3%	6.0%	8.7%	3.5%	9.4%	4.7%	7.8%
inconsistent	high high	14.7%	7.4%	16.8%	8.1%	18.4%	7.9%	16.6%	7.8%
	high low	13.2%	6.9%	15.7%	7.2%	19.8%	9.0%	16.2%	7.7%
	low high	14.9%	7.8%	17.2%	8.6%	16.9%	7.5%	16.3%	8.0%
	low low	8.9%	4.9%	13.7%	7.7%	11.7%	6.9%	11.4%	6.5%
semiconsistent	high high	13.4%	6.8%	11.4%	4.0%	8.8%	2.6%	11.2%	4.5%
	high low	8.5%	3.7%	10.9%	3.0%	9.4%	3.5%	9.6%	3.4%
	low high	13.7%	7.6%	11.4%	4.1%	9.3%	4.4%	11.5%	5.4%
	low low	7.3%	5.4%	9.5%	6.1%	5.7%	8.3%	7.5%	6.7%
	average	10.2%	5.8%	11.1%	6.3%	10.0%	6.8%		

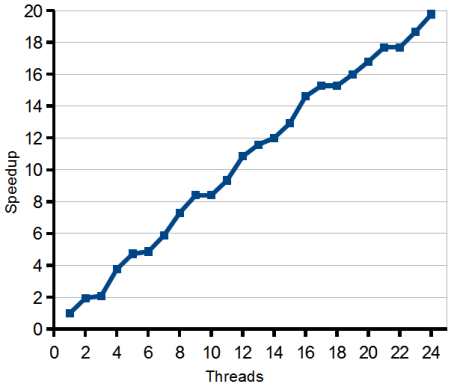


Figure 4: ME-MLS algorithm speedup.

The ME-MLS algorithm was evaluated over a large testbed of 792 instances with dimensions of up to 2048×64. The results were compared to four Min-Min based heuristics which tackle the energy-aware scheduling problem considering both minimization objectives. The experimental analysis showed that the ME-MLS method outperforms the Min-Min based heuristics with average improvements of up to **19.8%** for the makespan objective, and up to **9.4%** in the energy consumption objective. The performance analysis showed that the ME-MLS has a promising scalability behavior.

The main lines for future work include to improve the efficiency of the ME-MLS allowing us to solve larger problem dimensions. We also plan to extend the problem formulation to consider multicore machines and their energy consumption behaviors, this will allow us to model nowadays machine scenarios.

REFERENCES

Alba, E. and Luque, G. (2007). A new local search algorithm for the DNA fragment assembly problem. In *Proc. of 7^{th} European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 1–12.

Ali, S., Siegel, H., Maheswaran, M., Ali, S., and Hensgen, D. (2000). Task execution time modeling for heterogeneous computing systems. In *Proc. of the 9^{th} Heterogeneous Computing Workshop*, page 185, Washington, DC, USA.

Berman, F., Fox, G., and Hey, A. (2003). *Grid Computing: Making the Global Infrastructure a Reality*. Wiley.

Braun, T., Siegel, H., Beck, N., Bölöni, L., Maheswaran, M., Reuther, A., Robertson, J., Theys, M., Yao, B., Hensgen, D., and Freund, R. (2001). 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.

Dorronsoro, B., Bouvry, P., Cañero, J., Maciejewski, A., and Siegel, H. (2010). Multi-objective robust static mapping of independent tasks on grids. In *IEEE Congress on Evolutionary Computation*, pages 3389–3396.

El-Rewini, H., Lewis, T., and Ali, H. (1994). *Task scheduling in parallel and distributed systems*. Prentice-Hall, Inc.

Foster, I. and Kesselman, C. (1998). *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann.

Izakian, H., Abraham, A., and Snasel, V. (2009). Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. In *Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization - Volume 01*, CSO '09, pages 8–12, Washington, DC, USA. IEEE Computer Society.

Kessaci, Y., Mezmaz, M., Melab, N., Talbi, E., and Tuyttens, D. (2011). Parallel Evolutionary Algorithms for Energy Aware Scheduling. In *Intelligent Decision Systems in Large-Scale Distributed Environments*. Springer.

Khan, S. and Ahmad, I. (2009). 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.

Kim, J.-K., Siegel, H., Maciejewski, A., and Eigenmann, R. (2008). Dynamic resource management in energy constrained heterogeneous computing systems using voltage scaling. *IEEE Trans. Parallel Distrib. Syst.*, 19:1445–1457.

Kim, K., Buyya, R., and Kim, J. (2007). Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters. In *Proc. of the 7^{th} IEEE Int. Symposium on Cluster Computing and the Grid*, pages 541–548.

Kolodziej, J., Khan, S. U., and Xhafa, F. (2011). Genetic algorithms for energy-aware scheduling in computational grids. In *Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 17–24.

Lee, Y. and Zomaya, A. (2009). Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *Proc. of the 9^{th} IEEE/ACM Int. Symposium on Cluster Computing and the Grid*, pages 92–99.

Lee, Y. and Zomaya, A. (2011). Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Trans. Parallel Distrib. Syst.*, 22:1374–1381.

Leung, J., Kelly, L., and Anderson, J. (2004). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press.

Li, Y., Liu, Y., and Qian, D. (2009). A heuristic energy-aware scheduling algorithm for heterogeneous clusters. In *Proc. of the 2009 15^{th} Int. Conf. on Parallel and Distributed Systems*, pages 407–413.

Lindberg, P., Leingang, J., Lysaker, D., Khan, S., and Li, J. (2012). Comparison and analysis of eight scheduling heuristics for the optimization of energy consumption and makespan in large-scale distributed systems. *The Journal of Supercomputing*, 59(1):323–360.

Luo, P., Lü, K., and Shi, Z. (2007). 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.

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30.

Mezmaz, M., Melab, N., Kessaci, Y., Lee, Y., Talbi, E. G., Zomaya, A., and Tuyttens, D. (2011). A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *J. Parallel Distrib. Comput.*, 71(11):1497–1508.

Nesmachnow, S., Cancela, H., and Alba, E. (2010). Heterogeneous computing scheduling with evolutionary algorithms. *Soft Computing*, 15(4):685–698.

Nesmachnow, S., Cancela, H., and Alba, E. (2012). A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. *Appl. Soft Comput.*, 12(2):626–639.

Nesmachnow, S. and Iturriaga, S. (2011). Multiobjective scheduling on distributed heterogeneous computing and grid environments using a parallel micro-CHC evolutionary algorithm. In *Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 134–141.

Pecero, J., Bouvry, P., Fraire Huacuja, H. J., and Khan, S. (2011). 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*, pages 1–8.

Pinel, F., Dorronsoro, B., Pecero, J., Bouvry, P., and Khan, S. (2011). A two-phase heuristic for the energy-efficient scheduling of independent tasks on computational grids. *Journal of Cluster Computing*. Submitted.

AUTHOR BIOGRAPHIES

SANTIAGO ITURRIAGA has a degree on Engineering and is currently a student of the M.Sc. in Computer Science, Universidad de la Republica, Uruguay. He is an Assistant at Engineering Faculty, Universidad de la Republica. His M.Sc. thesis is focused on high performance computing and metaheuristics for scheduling. He has published several conference proceedings in these areas. Email: siturria@fing.edu.uy, personal webpage at http://www.fing.edu.uy/∼siturria.

SERGIO NESMACHNOW has a degree in Engineering (2000), a M.Sc. in Computer Science (2004), and a Ph.D in Computer Science (2010) from Universidad de la Republica, Uruguay. He is currently an Aggregate Professor at Numerical Computing Center, Engineering Faculty, Universidad de la Republica. His main research areas are scientific computing, high performance computing, and parallel metaheuristics, and their application for solving complex real-world problems. He has published over 50 papers in international journals and conference proceedings. He has also served in several technical program committees of international conferences in related areas, and he serves as reviewer for many journals and conferences. Email: sergion@fing.edu.uy, personal webpage at http://www.fing.edu.uy/∼sergion.

BERNABÉ DORRONSORO has a degree in Engineering (2002) and the Ph.D. in Computer Science (2007) from University of Málaga, Spain, and he is currently working as research associate at the University of Luxembourg. His main research interests include grid computing, ad hoc networks, the design of efficient metaheuristics, and their application for solving complex real-world problems in logistics, telecommunications, bioinformatics, combinatorial, multiobjective, and global optimization. He has several articles in impact journals and one book, has been member of the organizing committees of several conferences and workshops, and he usually serves as reviewer for leading impact journals and conferences. Email: beranbe.dorronsoro@uni.lu, personal webpage at www.bernabe.dorronsoro.es.