Energy Efficiency Management in Computational Grids through Energy-aware Scheduling

Silvana Teodoro, Andriele Busatto do Carmo, Luiz Gustavo Fernandes
GMAP Research Group (FACIN/PPGCC)
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Avenida Ipiranga, 6681 - Pr. 32, Porto Alegre, Brazil
{silvana.teodoro, andriele.carmo}@acad.pucrs.br, luiz.fernandes@pucrs.br

ABSTRACT

Energy consumption in High Performance Computing (HPC) has become an important issue in the past few years. The performance gain obtained by these environments is matched by a proportional increase of energy use. Example of such environments are computational grids, which are used in several academic and enterprise projects. Given this scenario, researchers have been trying to reduce the energy consumption while minimizing performance loss at the same time. This work proposes the use of energy-aware scheduling for energy efficiency management in computational grids. Our solution exploits the main existing approaches in the literature to reduce energy consumption in HPC environments: management of idle resources and energy-aware scheduling algorithms. We evaluate our proposed approach in a simulation environment and the algorithm was compared to other five traditional scheduling algorithms that do not consider energy features. Results show an energy reduction of up to 182.90% combined with a performance loss up to 27.78% in the best cases.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*cost estimation*; D.2.8 [**Software Engineering**]: Metrics—*performance measures, process metrics*; I.6.3 [**Computing Methodologies**]: Simulation and Modeling—*Applications*

General Terms

Algorithms, Experimentation, Management, Measurement

Keywords

Energy efficiency, scheduling algorithms, computational grid, simulation

1. INTRODUCTION

Energy efficiency in HPC has become an important issue in the past few years. High energy consumption leads to several environmental problems, and the performance gain often corresponds to higher energy consumption. HPC environments are thus targets for the development of solutions for the energy efficiency problem.

An example of a widely used high performance environment is a computational grid, since it allows the sharing of hardware and software resources transparently such as processing units, memory, persistent information, and others [12]. Such transparency implies that users should not be aware of heterogeneity, scalability and geographic location of resources. Furthermore, grids are environments with cheaper maintenance than others dedicated to HPC.

Recently, many works have been developed to solve the energy efficiency problem in HPC environments (see Section 2). Most of the approaches for reducing energy consumption are based either on turning off idle resources or scheduling tasks using specialized algorithms.

This work presents an approach to reduce the energy consumption of computational grids through energy-aware scheduling combined with a smart management of idle resources. Our main goal is to reduce the energy consumption for executing tasks while not decreasing significantly the overall application performance. The main contributions of this work can be summarized as follows:

- a general energy consumption model that can be used for any environment whose task sizes are accounted in *flops*; and
- a new energy-aware scheduling algorithm, that handles active hosts combined with a strategy to identify the right moment to turn off idle hosts.

To evaluate our approach, we simulate a grid environment using the SimGrid [1] framework along with LIBTS (Library Tasks Scheduling) [4]. For this work, we consider the most widely used kind of application for grid computing: bag-of-tasks applications [10] (*i. e.*, independent tasks with no communication among them). These applications are well suited for grids since in such environments the communication cost is quite significant due to the distribution of the hosts in a wide area network.

The rest of this paper is structured as follows: Section 2 presents the related work regarding energy efficiency in computational grids, classified according to their proposal in idle resources (Section 2.1) and scheduling (Section 2.2). Our approach is introduced in Section 3, describing in detail the simulation environment (Section 3.1), the consumption module (Section 3.2) and the proposed scheduling algorithm (Section 3.3). Section 4 presents the experimental setup used to run our experiments. Our simulation results are described in Section 5. Finally, Section 6 concludes the work with final remarks and considerations for future research.

SAC'13 March 18-22, 2013, Coimbra, Portugal.

2. RELATED WORK

Many works have been developed trying to reduce the energy consumption in computational grids. The most relevant to the best of our knowledge are presented here, and we classified them according to their approach. Some of these works manage idle hosts, turning them off when possible and the others schedule tasks following a specialized algorithm. Section 2.1 presents a review of approaches that manage idle hosts and Section 2.2 describes works based on scheduling algorithms.

2.1 Idle Resources Approach

Ponciano *et al.* [11] propose two strategies to save energy in computational grids:

- **standby** reduces the hard drive and processor activity. Just the RAM memory keeps working; and
- **hibernate** saves the memory RAM state at the hard drive reducing both memory and processor activity.

These strategies are applied whenever a host in a grid becomes idle. In this case, it is important to decide for how long the host must be idle in order to change its operation mode. According to the authors, it is important to choose the right time to apply one of the mentioned strategies. For example, if the time for sleeping is short, *i. e.*, if the host become inactive as soon as it become idle, the energy will be saved, but probably the time to respond to applications will be longer. On the other hand, if the inactivity time is long, the host will be for a long time on idle state consequently spending more energy, but the response time for applications will be shorter.

Another work, proposed by Olli Mämmelä *et al.* [9] presents a mechanism for turning off idle hosts in a grid. The time a task takes to start its execution is taken into account to decide when turn off hosts. This work also presents models to determine the energy consumption of several computer components whenever they are idle, such as processors, memory, hard drive, and others. Tasks with different behavior were used to run tests, allowing the stressing of different machine components (CPU, memory, and others).

2.2 Scheduling Approach

Lizhe Wang *et al.* [14] propose a study to predict a whole data center temperature. According to them, high temperatures can increase the cooling cost and also increase the frequency of hardware failures. As it is possible to reduce the hosts temperature managing their load, the authors propose a temperature prediction scheduling algorithm using an Artificial Neural Network (ANN), named Thermal-Aware task Scheduling Algorithm (TASA). This algorithm aims to reduce the energy and temperature consumption in a data center.

The work developed by Saurabh Kumar Garg *et al.* [5] proposes a scheduling algorithm that distributes applications in a grid. This distribution uses the Heterogeneity Aware Meta-scheduling Algorithm (HAMA) that selects grid resources according to their energy efficiency, from the most to the less efficient. HAMA sorts received tasks using as base the Earliest Deadline First (EDF) algorithm [8]. After that, it uses the Dynamic Voltage Scaling (DVS) technique to change the frequency of CPU to achieve a more significant reduction of energy consumption. It is important to highlight that HAMA respects the deadline to the end of the tasks execution.

The work proposed by Michael Lammie *et al.* [7], is a scheduler aiming at reducing energy consumption in a cluster while preserving some performance. To achieve this, the authors attempted to change the number and frequency of available processors. They have proposed three strategies:

- CPU frequency scaling - a technique to switch the CPU frequency aiming at reducing the the energy consumption and dissipate the heat;
- automatically sizing hosts - it allows to turn on/off hosts aiming at attending some requirements, such as queue size and tasks characteristics. Hosts are only turned on when the active ones do not support the overall load, and they are turned off after being idle for a specific amount of time; and
- smart jobs allocation - optimization layer implemented in cluster management algorithms, that looks to the environment seeking for active hosts that are idle.

Even though these works try to improve the energy efficiency by using scheduling approaches, they differ from ours in some aspects. Firstly, our solution is oriented to tasks that do not present deadlines and are totally independent from each other (*i. e.*, there is no communication among them). Secondly, our approach is specific for computational grids differently from the work of [7], and consequently our scheduling algorithm is framed by some specific aspects of those platforms. For instance, in such platforms it is not always possible to modify the CPU frequency of the hosts.

3. ENERGY-AWARE SCHEDULING

This section presents our approach based in a consumption module and an energy-aware scheduling algorithm. First, we introduce the simulation environment in Section 3.1. Then, in Section 3.2, we present our consumption module detailing the required information to compute the energy spent while executing an application. Finally, our energy-aware scheduling algorithm (named Low Energy Consumption Scheduling Algorithm – LECSA) is discussed in Section 3.3.

3.1 Simulation Environment

This section presents the simulation environment we used for test our approach for managing energy efficiency in computational grids. Ideally, it would be better to carry out our experiments on a real grid environment, since we would have more reliable results representing the real system behavior. However, in grid computing this is quite rare because building and maintaining a grid infrastructure is timely and financially expensive. Thus, it is not realistic to use such environments for experiments at anytime. In this scenario, simulate real environments behavior becomes an interesting alternative, since we do not need to allocate a real infrastructure to perform experiments. Moreover, with simulation it is possible to run experiments and combine different scenarios in a shorter time [2].

We chose SimGrid [1] and LIBTS [4] as the tools to set our experimental environment. SimGrid is a distributed systems simulation framework and LIBTS is a library implemented using the SimGrid API (version 3.5) which simulates several classical scheduling algorithms. As illustrated in Figure 1, LIBTS is implemented inside the MSG SimGrid module. Examples of scheduling algorithms available in LIBTS are: WQ (Work Queue), WQR (Work Queue with Replication), Sufferage, XSufferage and Dynamic FPLTF (Fastest Processor to Largest Task First). We integrated our consumption module in this architecture in order to collect the energy spent by each algorithm during the runtime. Moreover, LECSA was added to the set of scheduling algorithms offered by LIBTS.

3.2 Consumption Module

In order to estimate the energy consumption of an application and then know how much energy was spent for each scheduling algorithm, we created a simple and efficient

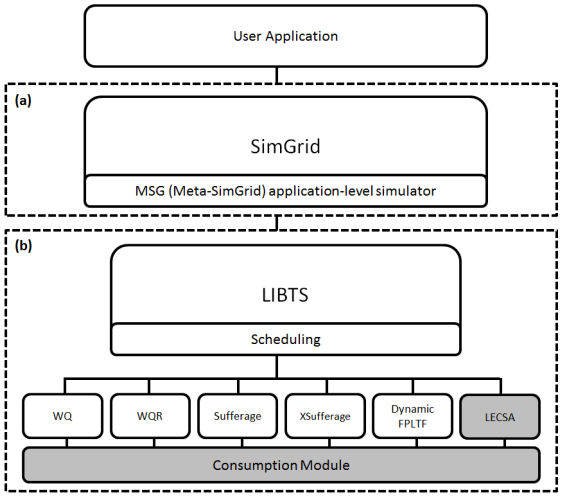


Figure 1: SimGrid Development Environment: SimGrid Module MSG (a); LIBTS Architecture (b).

consumption module. This module computes each task and each host energy consumption, as well as the total energy spent during the application execution. This module requires three basic information to work properly: (i) the task size in *flops*, (ii) the host energy-efficiency in *flops/watt* and (iii) the amount of energy spent when the host is idle. This module can be adapted for any simulation environment with the same model of power and consumption measures.

Firstly, it is necessary to know how much energy is consumed by a single task. Equation (1) computes how much energy is consumed during a task execution, where: $taskSize$ represents the task size and $energyEfficiency$ is a measure to point out how many *flops* can be executed while a single *watt* is spent (*flops/watt*). This measurement is used in several researches, including the Top500 list [13]. Thus, the energy consumption is given by:

$$ecTask = taskSize/energyEfficiency. \quad (1)$$

Since we know the energy spent by each task during its execution, it is possible to know how much energy a host spends to execute all tasks assigned to it. This value can be computed through Equation (2), where: $\sum ecTask_i$ is the sum of each task energy consumption executed in a specific host; $timeIdle$ is the idle time of the host; and $ecIdle$ is the host energy consumption on idle mode. In this case, we add to the sum of all tasks energy consumption the idle time of the host multiplied by its idle consumption. The energy consumption of each host is computed through:

$$ecHost_i = \sum ecTask_i + (timeIdle * ecIdle). \quad (2)$$

Finally, in order to know the total energy consumption of the environment, Equation 3 presents the sum of the energy consumed by all the hosts during their idle and active time:

$$ecTotal = \sum ecHost_i. \quad (3)$$

Thus, it is possible to know how much energy was spent while an application was executing according to the used scheduling algorithm.

3.3 Scheduling Algorithm

Aiming at managing the energy efficiency in a computational grid, we developed an energy-aware scheduling algorithm named LECSA. In order to schedule tasks, LECSA presents four well-defined steps:

1. tasks are sorted in descending order, according to $taskSize$. Task size is measured in *flops*;
2. hosts are also sorted in descending order, according to their energy-efficiency value which indicates how many *flops* can be executed with a single *watt*;
3. it is decided how many and what hosts will be used during tasks execution. This decision is based on the energy-efficiency (*flops/watt*) of each host, the amount of tasks to be computed and the tasks size heterogeneity (the tasks size heterogeneity variation is described in Section 4). There are two possible scenarios (in both, idle hosts are turned off aiming at reducing the energy consumption):
   - the amount of tasks is smaller than the number of available hosts - in this case, only hosts having energy-efficiency values among the top 30% are used;
   - the amount of tasks is larger than the number of available hosts - here, the decision is based on tasks heterogeneity. The higher is the tasks size heterogeneity, the higher is the amount of hosts with different energy-efficiency values used. For instance, for tasks with 0% heterogeneity, only the hosts having energy-efficiency among the top 20% are used. With tasks size heterogeneity varying up to 25%, the hosts having energy-efficiency among the top 40% are used and so on. The entire set of hosts is used only when tasks size heterogeneity varies up to 100%.
4. tasks are divided among all hosts defined in step 3. The goal is to send the same number of tasks to each host. When the result of this division (total number of tasks by the total number of active hosts) is not exact, each left task is sent to the first host of the list (which has the higher energy-efficiency). Notice that tasks are sent to hosts in a sequence of different rounds. There will be as many rounds as the total number of tasks divided by the number of active hosts. One task at a time is sent to each host, keeping all hosts occupied during the distribution.

Table 1: Application Granularity.

Amount of tasks	Hosts/tasks	Task size (*flops*)
720	0.125	1,000,000
144	0.625	5,000,000
29	3.103	25,000,000

4. EXPERIMENTAL SETUP

This section describes our experimental setup. We first explain the granularity of applications. Then, we introduce the hosts and tasks heterogeneity and finally we present the hosts energy-efficiency configuration.

As mentioned in Section 3, our simulation uses the SimGrid framework [1]. We chose this simulation framework due to its compatibility with our model, where computational power is represented by *flops* and the energy-efficiency can be handled in *flops/watt*. In our tests, we use WQ, WQR, Sufferage, XSufferage and Dynamic FPLTF since they are scheduling algorithms specialized for grids. We configured 90 hosts in the simulated computational grid.

Table 2: Hosts Configuration.

Base hosts				Simulation hosts	
Green500 list	Top500 list	Power (*Teraflops*)	Energy-efficiency (*Megaflops/watt*)	Power (*flops*)	Energy-efficiency (*flops/watt*)
1	64	172.49	2,026.48	6,000	0.070
58	228	72.03	467.73	7,000	0.045
7	114	103.20	1,266.26	8,000	0.098
40	235	70.28	731.85	9,000	0.094
110	477	51.88	341.32	10,000	0.066
123	51	194.40	318.69	11,000	0.018
127	217	73.83	305.33	12,000	0.049
100	366	56.73	354.56	13,000	0.081
82	62	174.90	372.02	14,000	0.029

Granularity of Applications

We defined the application granularity based on the work of [3] which sets the basis for constructing test scenarios for grid computing experiments. Therefore, we set up 3 groups composed by applications divided in different amounts of tasks: 29, 144 and 720. For each group, there is a task mean size (see Table 1). Our goal was to configure different scenarios, testing situations in which the number of tasks is much smaller than, close to and much bigger than the number of hosts.

Hosts Heterogeneity

We used two environments with different levels of heterogeneity among their hosts in order to reproduce as close as possible the configuration of real computational grids. In this sense, our grid machines present a computational power measured in *flops* and this power respects a specific limit. Each host speed is defined according to a uniform distribution keeping the average of all hosts speed which is nearby 10,000 *flops*. The levels of heterogeneity we used are:

- **low heterogeneity** - the hosts speed diversify according to an uniform distribution from 9,000 up to 11,000; and
- **high heterogeneity** - the hosts speed diversify according to uniform distribution from 6,000 up to 14,000.

Tasks Heterogeneity

We also defined different task sizes, from homogeneous (0%) to completely heterogeneous (100%). The ranges used were 0%, 25%, 50%, 75% and 100% of the mean size of tasks in the group. For example, for a range of 25%, tasks size can present values up to 12.5% below and above the mean size (1,000,000; 5,000,000 and 25,000,000).

Hosts Energy-efficiency Configuration

Find the power in *flops* and energy-efficiency in *flops/watt* of off-the-shelf machines is not a trivial task. For that reason, the values of hosts energy-efficiency for our simulation were defined proportionally to real machines configuration (extracted from Green500 [6] and Top500 [13] lists from November, 2011). Table 2 shows all values used to compute the energy-efficiency of each simulated host according to its computational power. We chose the hosts according to their characteristics, trying to keep our simulated environment more realistic. Their ranking in Green500 and Top500 lists are placed at the first columns followed by their computational power and energy-efficiency values.

Our machines have computational power ranging from 6,000 up to 14,000 *flops*. The energy consumption values of idle hosts was defined randomly (from 4 up to 12 *watts* per hour). These values are usually lower than the energy consumed during any task execution.

5. SIMULATION RESULTS

In this section, we present the obtained results in our simulation experiments in terms of energy consumption and performance gain/loss for each environment previously introduced. Figure 2 allows a visual comparison of the consumption and performance results of each tested scheduling algorithm.

In Table 3, we can clearly identify in which cases LECSA presented the best and worst energy efficiency and performance as well. LECSA saved 182.9% of energy in Environment 2 with 29 homogeneous tasks with a performance loss equivalent to 27,78%. The lower energy consumption reduction occurs in Environment 1 with 720 tasks and 25% of tasks size range. In this particular case, the performance gain was 0.27%.

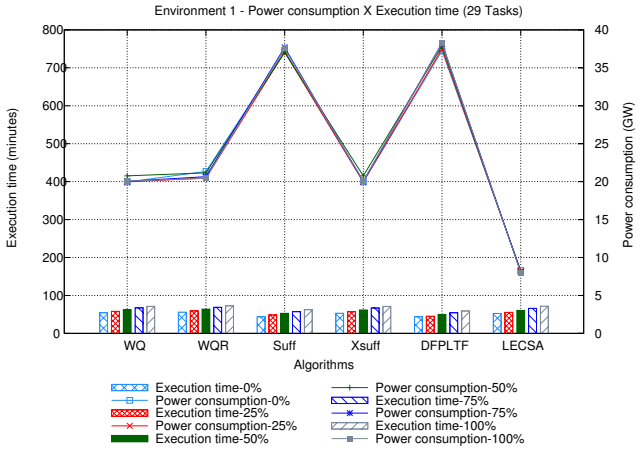
In terms of performance, the maximum gain achieved by LECSA was 9.21% in Environment 1 with 144 100% heterogeneous tasks with a reduction in consumption equivalent to 50.66%. On the other hand, the maximum loss of performance was 54.75% in Environment 2 with 144 homogeneous tasks with 102.78% energy efficiency improvement. Also, it is possible to notice that in scenarios with 29 tasks (much more machines than tasks) either for Environments 1 or 2, LECSA presented the best results for energy efficiency.

We realize that the higher is the hosts heterogeneity, the higher is the reduction of consumption obtained using LECSA. However, the performance decreases as well. Also, the higher is the granularity of application (29 tasks with size of 25,000,000 *flops*), the higher is the energy saved (*e. g.*, 147.57% in average for Environment 1). In this case, the performance loss is 38.25% in average. On the other hand, the lower granularity (720 tasks with size of 1,000,000 *flops*) presents the smallest energy economy (at most 59.04% in average) with a performance loss about 1.82% in average.

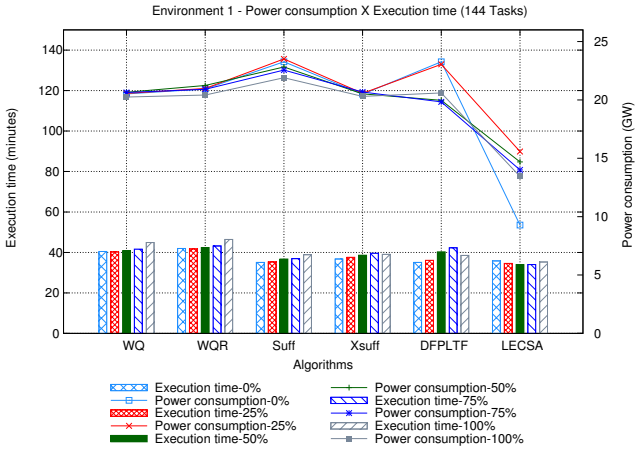
Finally, in terms of tasks heterogeneity, LECSA presented an energy consumption reduction between 100.00% and 182.90% for homogeneous tasks. On the opposite (100% heterogeneity), the consumption gain ranged from 23.20% up to 150.24%.

6. CONCLUDING REMARKS

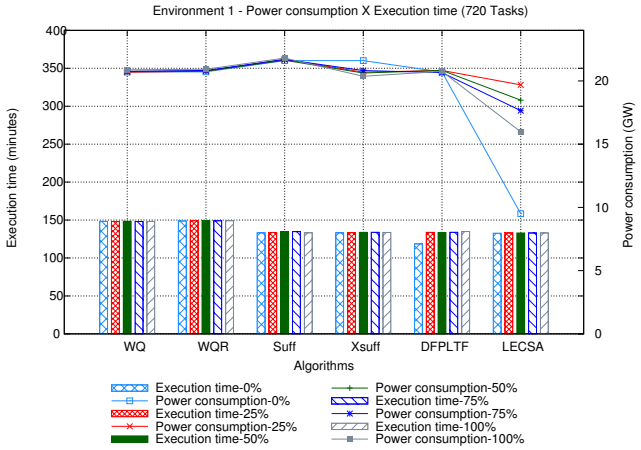
This paper presented an energy efficiency management solution for computational grids using energy-aware scheduling. This solution is based on a scheduling algorithm named LECSA which tries to assign higher energy consumption tasks to hosts with better energy-efficiency values turning off unused hosts when necessary. We compared our algorithm to several classical scheduling algorithms in terms of energy consumption and performance. Naturally, LECSA



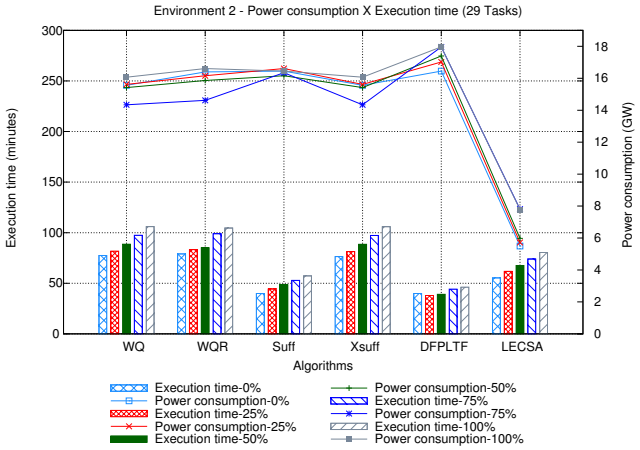
(a) Environment 1 with 29 tasks



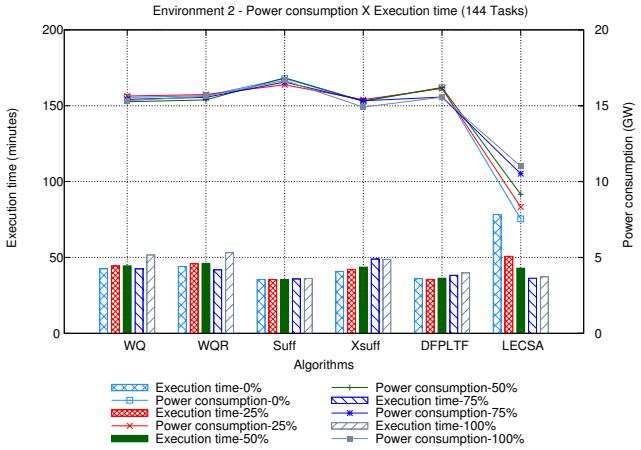
(b) Environment 1 with 144 tasks



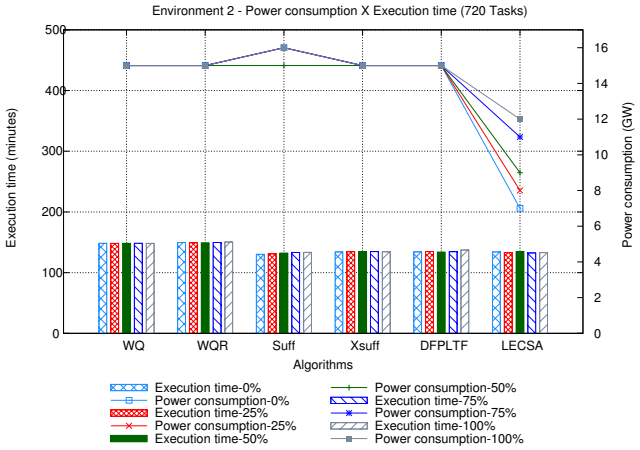
(c) Environment 1 with 720 tasks



(d) Environment 2 with 29 tasks



(e) Environment 2 with 144 tasks



(f) Environment 2 with 720 tasks

Figure 2: Simulation Results

presented in all cases the best energy reduction in comparison to classical algorithms since their main goal is to increase the performance gain regardless of energy consumption concerns. In general, LECSA presented some loss of performance in comparison to the others algorithms which is acceptable if a reduction in energy consumption is possible. In many cases, however, its performance was close to the best classical algorithm. Finally, it is worthy to highlight that in some cases, LECSA surprisingly achieved performance improvement as well.

Table 3: LECSA Performance and Consumption loss/gain.

			Consumption				Performance			
	#Tasks	Heterogeneity	Best	2°	Rank LECSA	Diff. % LECSA	Best	2°	Rank LECSA	Diff. % LECSA
Environment 1	29	0%	LECSA	XSuff	1°	+141.06	XSuff	Suff	3°	-16.64
		25%	LECSA	XSuff	1°	+143.23	DFPLTF	Suff	3°	-17.93
		50%	LECSA	XSuff	1°	+155.03	DFPLTF	Suff	3°	-17.81
		75%	LECSA	XSuff	1°	+148.32	DFPLTF	Suff	3°	-17.92
		100%	LECSA	XSuff	1°	+150.24	DFPLTF	Suff	5°	-17.94
	144	0%	LECSA	XSuff	1°	+120.79	DFPLTF	Suff	3°	-1.95
		25%	LECSA	WQ	1°	+31.96	LECSA	Suff	1°	+2.16
		50%	LECSA	DFPLTF	1°	+35.94	LECSA	Suff	1°	+7.99
		75%	LECSA	DFPLTF	1°	+41.64	LECSA	Suff	1°	+8.58
		100%	LECSA	WQ	1°	+50.66	LECSA	DFPLTF	1°	+9.21
	720	0%	LECSA	DFPLTF	1°	+118.00	DFPLTF	LECSA	2°	-10.70
		25%	LECSA	XSuff	1°	+5.07	LECSA	XSuff	1°	+0.27
		50%	LECSA	XSuff	1°	+11.63	LECSA	DFPLTF	1°	+0.45
		75%	LECSA	DFPLTF	1°	+17.05	LECSA	DFPLTF	1°	+0.72
		100%	LECSA	Suff	1°	+27.47	LECSA	Suff	1°	+0.14
Environment 2	29	0%	LECSA	XSuff	1°	+182.90	DFPLTF	Suff	3°	-27.78
		25%	LECSA	XSuff	1°	+173.07	DFPLTF	Suff	3°	-38.58
		50%	LECSA	XSuff	1°	+157.85	DFPLTF	Suff	3°	-42.12
		75%	LECSA	XSuff	1°	+83.61	DFPLTF	Suff	3°	-40.17
		100%	LECSA	XSuff	1°	+106.55	DFPLTF	Suff	3°	-42.60
	144	0%	LECSA	XSuff	1°	+102.78	Suff	DFPLTF	6°	-54.75
		25%	LECSA	XSuff	1°	+84.87	DFPLTF	Suff	6°	-30.33
		50%	LECSA	WQ	1°	+66.59	Suff	DFPLTF	3°	-17.42
		75%	LECSA	XSuff	1°	+45.72	Suff	LECSA	2°	-1.34
		100%	LECSA	XSuff	1°	+35.54	Suff	LECSA	2°	-3.11
	720	0%	LECSA	XSuff	1°	+100.00	Suff	LECSA	2°	-2.73
		25%	LECSA	XSuff	1°	+77.86	Suff	LECSA	2°	-1.55
		50%	LECSA	XSuff	1°	+59.42	Suff	DFPLTF	4°	-1.71
		75%	LECSA	DFPLTF	1°	+34.72	LECSA	Suff	1°	+0.67
		100%	LECSA	XSuff	1°	+23.20	LECSA	Suff	1°	+0.39

As future works, we believe that our solution can be improved taking into account the energy spent by communication operations among tasks. Also, we believe that a dynamic solution based on specialized energy-aware scheduling algorithms is the natural path to follow. In this case, the environment should be capable of choosing the best scheduling algorithm depending of the input tasks configuration.

Acknowledgements

This work is part of a research project with financial support of Brazilian research agencies: Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) and Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS).

7. REFERENCES

[1] H. Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *CCGRID*, pages 430–441. IEEE Computer Society, 2001.

[2] H. Casanova, A. Legrand, and M. Quinson. Simgrid: A generic framework for large-scale distributed experiments. In D. Al-Dabass, A. Orsoni, A. Brentnall, A. Abraham, and R. N. Zobel, editors, *UKSim*, pages 126–131. IEEE, 2008.

[3] D. P. da Silva, W. Cirne, and F. V. Brasileiro. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In H. Kosch, L. Böszörményi, and H. Hellwagner, editors, *Euro-Par*, volume 2790 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2003.

[4] P. B. Franco. Escalonamento de tarefas em ambiente de simulação de grid computacional. Master's thesis, Universidade Estadual Paulista, 2011.

[5] S. K. Garg and R. Buyya. Exploiting heterogeneity in grid computing for energy-efficient resource allocation. In *The 17th International Conference on Advanced Computing and Communications*, 2009.

[6] Green500. The green500. http://www.green500.org/, November 2011.

[7] M. Lammie, P. Brenner, and D. Thain. Scheduling grid workloads on multicore clusters to minimize energy and maximize performance. In *GRID*, pages 145–152. IEEE, 2009.

[8] J. Y.-T. Leung and J. H. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[9] O. Mämmelä, M. Majanen, R. Basmadjian, H. D. Meer, A. Giesler, and W. Homberg. Energy-aware job scheduler for high-performance computing, 2011.

[10] M. A. S. Netto and R. Buyya. Coordinated rescheduling of bag-of-tasks for executions on multiple resource providers. *Concurrency and Computation: Practice and Experience*, 24(12):1362–1376, 2012.

[11] L. Ponciano and F. V. Brasileiro. On the impact of energy-saving strategies in opportunistic grids. In *GRID*, pages 282–289. IEEE, 2010.

[12] U. Schwiegelshohn, R. M. Badia, M. T. Bubak, M. Danelutto, S. Dustdar, F. Gagliardi, A. Geiger, L. Hluchy, D. Kranzlmüller, E. Laure, T. Priol, A. Reinefeld, M. Resch, A. Reuter, O. Rienhoff, T. Rüter, P. M. A. Sloot, D. Talia, K. Ullmann, R. Yahyapour, and G. von Voigt. Perspectives on grid computing. *Future Generation Computer Systems*, 26(8):1104–1115, 2010.

[13] Top500. Top500 supercomputers site. http://www.top500.org, November 2011.

[14] L. Wang, G. von Laszewski, F. Huang, J. Dayal, T. Frulani, and G. Fox. Task scheduling with ANN-based temperature prediction in a data center: a simulation-based study. *Eng. Comput. (Lond.)*, 27(4):381–391, 2011.