An Adaptive Approach to Task Scheduling Optimization in Dynamic Grid Environments

J. Díaz, C. Muñoz-Caro and A. Niño
Grupo de Química Computacional y Computación de Alto Rendimiento,
Escuela Superior de Informática, Universidad de Castilla-La Mancha,
Paseo de la Universidad 4, 13071, Ciudad Real, Spain

Abstract— *Scheduling algorithms play an important role in heterogeneous computing systems. These algorithms guarantee a good load balance, also reducing the communication overhead. In particular, Quadratic Self-Scheduling (QSS) and Exponential Self-Scheduling (ESS) are flexible enough to adapt to distributed systems, thanks to their adjustable parameters. This flexibility becomes very important when the environment is dynamic because only algorithms that can shape the workload will be able to handle dynamic environment such as an Internet-based Grids of computers. In this sense, we propose a new adaptive approach that allows to obtain the optimal parameters of QSS and ESS each time that the properties of the environment change. We find that using this approach, the QSS algorithm improves its performance around a 25%. However, it is shown that the performance of ESS decreases when there is a large number of tasks. Finally, the overall time spent by the adaptive approach to generate new optimal sets of parameters is just a bit higher than the time to obtain just once. However, it is not proportional to the number of new sets of parameters generated. This is because the number of tasks decreases each time that we obtain new optimal parameters.*

Keywords: Self-Scheduling Algorithms, Adaptive Scheduling, Computational Grid

1. Introduction

Computational Grids [1] provide an opportunity to share a large number of resources among different organizations in an unified way, maximizing their use. A Grid can be used effectively to support large-scale runs of distributed applications. An ideal case to be run in Grid is that with many large independent tasks. This case arises naturally in parameter sweep problems. A correct assignment of tasks, so that computer loads and communication overheads are well balanced, is the way to minimize the overall computing time. This problem belongs to the active research topic of the development and analysis of scheduling algorithms. Different scheduling strategies have been developed along the years (for the classical taxonomy see [2]). In particular, dynamic self-scheduling algorithms are extensively used in practical applications. These algorithms represent adaptive schemes where tasks are allocated in run-time. Self-scheduling algorithms were initially developed to solve parallel loop scheduling problems in homogeneous memory-shared systems, see for instance [3]. This kind of algorithms divides the set of tasks into subsets (chunks), and allocates them among the processors. In this way overheads are reduced.

Although self-scheduling algorithms were derived for homogeneous systems, they could be applied to heterogeneous ones such as computational Grids [4],[5],[6],[7]. However, the problem could be the flexibility of these algorithms (they may have not enough degrees of freedom) to adapt efficiently to a heterogeneous environment. Thus, we have previously proposed two new flexible self-scheduling algorithms called Quadratic Self-Scheduling (QSS) [8],[6] and Exponential Self-Scheduling (ESS) [7],[9]. The first is based on a quadratic form for the chunks distribution function. Therefore, it has three degrees of freedom, which provide high adaptability to distributed heterogeneous systems. The second approach, ESS, is based on the slope of the chunks distribution function. In this case, we consider that the chunks distribution function decreases in an exponential way. This algorithm provides a good adaptability in distributed heterogeneous systems using two parameters. Moreover, in previous works [7] we have compared our approaches with other self-scheduling algorithms in an actual Grid environment. The new algorithms outperform the previous ones, since they obtain better load balance and more reduction of the communication overhead [7].

However, a computational Grid is made up of a large number of independent resource providers and consumers, which run concurrently, change dynamically, and interact with each other. Due to these environment characteristics, new approaches such as those based in heuristic algorithms [10],[11] have been proposed to address the challenges of Grid computing. These kinds of algorithms make realistic assumptions based on a priori knowledge of the concerning processes and of the system load characteristics. Braun et al [12] presented three basic heuristics, based on Nature, for Grid scheduling. These are Genetic Algorithms (GA), Simulated Annealing (SA) and Tabu Search (TS).

Genetic algorithms (GA) [13] provide a robust searching technique that allows a high-quality solution to be derived from a large search space. This solution is obtained in

polynomial time by applying the principle of evolution. One of the different uses of these algorithms is in Grid Scheduling as seen in [14], [15]. Another heuristic algorithm is Simulated Annealing (SA) [16]. SA derives from the Monte Carlo method for statistically searching global minima. The method arises from a thermodynamic analogy. Specifically, it simulates the way that liquids freeze and crystallize, or metals cool and anneal. Here, we focus in SA for scheduling purposes. This approach has been tested in different environments like computational Grids [17],[18]. In Tabu Search (TS) [19] some historical information related to the evolution of the search is kept. This is basically the itinerary through the visited solutions. Such information is used to guide the movement from one solution to the next, avoiding cycling. We can found in [20] the use of this heuristic to schedule workflows for computational Grids.

Other way to tackle the problem of the unpredictable changing conditions of the Grid is using Adaptive Grid scheduling [21]. In general, adaptive scheduling can consider, among others, factors such as availability, performance, workload, bandwidth and latency of the interconnection links, etc, properly scaled according to the application needs. In this sense, there are several ways to provide adaptive Grid scheduling. Thus, for instance, we can find frameworks as AppLeS [22], Nimrod/G [23], Cactus Worm [24] or GridWay [21]. Moreover, there are algorithms like adaptive factoring [25] [26] that estimate loop iterations execution times and, using an statistical model, allocate dynamically chunks of tasks among processors. Other examples of adaptive scheduling could be the algorithms Sufferage [27] and XSufferage [28] which assign tasks to processors by considering the minimum completion time (minimum time to obtain the result of a task), at the host level and at the cluster level, respectively.

As presented above, the QSS and ESS self-scheduling algorithms depend on three and two parameters respectively. These parameters determine the behavior of the algorithms. Therefore, for a given computational environment it is necessary to select the most appropriate (optimal) values of these parameters to obtain a good load balance and to minimize the overall computation time. Previously, we presented a way to obtain optimal QSS and ESS parameters using a heuristic approach [29]. To such an end, we simulate the execution environment (a computational Grid in our case). In this work, we consider that the Grid is a dynamic environment and therefore it changes over time. Thus, using the simulation, we can obtain the computational time of each algorithm for a given value of its parameters. Therefore, it is possible to apply an heuristic algorithm to explore the behavior of the scheduling method for different values of the parameters, minimizing the overall computation time. The heuristic algorithm selected was Simulated Annealing (SA). However, these parameters are the "best" ones for the environment where they were obtained. Therefore, if the environment characteristics change strongly, we could have a lack of efficiency.

In this paper, we present an adaptive approach to scheduling tasks in a Grid environment. Thus, when the environment changes, new optimal QSS and ESS parameters are obtained. In this way, it is possible to maintain a good load balance. We have used this approach to compare the performance of the QSS and ESS algorithms.

In the next section, we present an overview of the QSS and ESS self-scheduling algorithms, as well as the methodology used for their optimization. Section 3 presents and interprets the results found in the optimization process. Finally, in section 4 we present the main conclusions of this paper and the perspectives for future works.

2. Methodology

In this work the Quadratic Self-Scheduling (QSS) and Exponential Self-Scheduling (QSS) algorithms are used as basic scheduling strategies. QSS [6][7][9] is based on a Taylor expansion of the chunks distribution function, C(t), limited to the quadratic term. Therefore, QSS is given by

$$C(t) = a + bt + ct^2 \tag{1}$$

where t represents the t-th chunk assigned to a processor. To apply QSS we need the a, b and c coefficients of equation (1). Thus, assuming that the quadratic form is a good approach to C(t), we can select three reference points (C(t), t) and solve for the resulting system of equations. Useful points are $(C_0, 0)$, $(C_{N/2}$, N/2) and (C_N, N), where N is the total number of chunks. Solving for a, b and c, we obtain,

$$\begin{aligned} a &= C_0 \\ b &= (4C_{N/2} - C_N - 3C_0)/N \\ c &= (2C_0 + 2C_N - 4C_{N/2})/N^2 \end{aligned} \tag{2}$$

where N is defined [6] by,

$$N = 6I/(4C_{N/2} + C_N + C_0) \tag{3}$$

being I the total number of tasks.

The $C_{N/2}$ value is given by,

$$C_{N/2} = \frac{C_N + C_0}{\delta} \tag{4}$$

where δ is a parameter. Assuming C_0 and C_N fixed, the $C_{N/2}$ value determines the slope of equation (1) at a given point. Therefore, depending on δ, the slope of the quadratic function for a t value is higher or smaller than that of the linear case, which corresponds to δ=2. In the present work, the values of the three parameters, C_0, C_N and δ are heuristically optimized in the simulated execution environment.

On the other hand, we have Exponential Self-Scheduling (ESS) [7][9]. ESS belongs to the family of algorithms based in the slope of the chunks distribution function, C(t). So, we consider that the rate of variation of C(t) is a decreasing function of t, g(t). Therefore, we have the general expression,

$$\frac{dC(t)}{dt} = g(t) \tag{5}$$

Equation (5) defines a differential equation. After integration we will have an explicit functional form for C(t) as a function of t.

ESS considers that the slope (negative) is proportional to the chunk size,

$$\frac{dC(t)}{dt} = -kC(t) \tag{6}$$

Here, k is a parameter and t represents the t-th chunk assigned to a processor. Equation (6) can be integrated by separation of variables yielding [7],

$$C(t) = C_0 e^{-kt} \tag{7}$$

Equation (7) defines the Exponential Self-Scheduling (ESS) algorithm. Here, C_0 and k are the parameters to be optimized in the simulated working environment.

With respect to the SA heuristic approach [29], as applied here, we consider that the function to minimize, the cost function (f), is the overall computation time needed to process a set of tasks, i.e., its makespan. In turn, we consider that the cost function depends on s, the set of parameters used by each self-scheduling algorithm.

The cost function f(s) is obtained as the simulated time (in seconds) necessary to solve all tasks in the specified execution environment. The tasks are scheduled according to QSS or ESS, and the optimal parameters are given by the final value of s obtained by simulated annealing. The simulator is organized as follows. Each task has associated a value, from 1 to 10, which represents its duration (in seconds). Task durations are randomly generated. As previously commented, QSS and ESS allocate sets of tasks (chunks). Therefore, the duration of a chunk is the sum of all tasks durations composing it. The computing (CPU) time for a chunk is calculated dividing its duration by the relative computing power of the processor where the chunk is executed. This computing power is referred to the fastest processor. Thus, lower values correspond to slower processors. To this value we add the temporal cost of transferring the chunk to the processor where it is executed. In addition, the scheduling cost introduced by the local queuing software is included as well.

The execution environment represented in the simulation is an Internet-Based Grid of computers [1]. Therefore, it is composed by two main components: network links and computer elements. Network links have associated a value that represents the temporal cost, in seconds, of transferring a file between two machines through Internet. This value is obtained as an approximate estimate of the transport latency [30]. To such an end, we consider actual measures of the bandwidth (bw) and of the smoothed round trip time (srtt) [31]. This last estimates future round trip times by sampling the behavior of packets sent over a connection and averaging those samples. Half the srtt is used as a rough estimate of an effective time of flight. So, the temporal cost of a network link (Tt) is given by $Tt = (file\_size/bw) + srtt/2$, where "file_size" represents the physical size of the chunk being sent. We consider that all chunks have the same physical size. On the other hand, a computer element is typically a computer cluster [32]. This is composed by a number of processors connected through an internal network. So, each simulated computer element has an associated array, which collects the relative computing power (a real number) for its processors (CPUs). The computer power of each processor is determined experimentally. The temporal cost of the internal network is considered negligible with respect to the temporal cost of the Internet network links.

Using the previous heuristic approach, we can obtain the "best" parameters for a given environment. However, if the environment characteristics change strongly we could have a lack of efficiency. Therefore, in this work we present an adaptive approach to tackle this case. Thus, we have created an application, which allows to simulate the execution of the chunks (step by step) in an environment as the one presented before. Moreover, if the environment changes, it is updated in the simulator, generating a new list of chunks to continue with the execution. The previous heuristic will be used to generate a new list of chunks. The corresponding pseudocode is shown in Chart 1.

Chart 1: Pseudocode for the Adaptive Approach

```
Function AdaptiveApproach()
    1.-Generate simulated execution
       environment
    2.-Obtain optimal parameters
    3.-Generate list of chunks
    while there are chunks do
      4.-Start/Continue simulation of chunks
      if environment changes then
        5.-Save current status
        6.-Update execution environment
        7.-Obtain new optimal parameters
        8.-Generate new list of chunks
      end_if
    done_while
end
```

The first step generates the simulated execution environment. After that, the optimal parameters of QSS or ESS algorithms are obtained using the heuristic approach [29] described before. With these parameters, we generate the list of chunks to schedule the tasks. From here, the "while" loop represents the new adaptive approach. In step 4, we simulate the execution of chunks in the previous execution

environment. If the environment changes, the application will have to adapt itself. Therefore, the current execution status is saved (step 5). This information is a Gantt chart [33] of the estimated completion time, in each processor, of the chunks allocated. Now, we update the execution environment characteristics and restore the saved execution status. Since the environment has changed, we have to generate new optimal parameters for our self-scheduling algorithm. For this, we use the heuristic approach presented in [29]. To obtain new parameters we have to take into account the saved execution status to consider only the non-allocated tasks. This is important because each processor has a different estimated completion time for the chunks assigned. In this way, we can guarantee a better load balance. Once we have the new optimal parameters for the self-scheduling algorithms, we generate a new list of chunks going back to the 4th step. The "while" loop finishes when all tasks are scheduled.

The simulated execution environment used is a replica of the computational Grid considered in [7]. The Grid is made up by a client (qcycar) and three computer elements (C.E.): Hermes, Tales and Popocatepetl (Popo). Hermes and Tales are placed in Ciudad Real (Spain), and they are composed by 8 processors each. On the other hand, Popo is placed in Puebla (Mexico), and it is composed by 4 processors. We have subdivided the environment characteristics into three parts: network, processors and scheduling cost. The network characteristics, for each computer element, are collected in Table 1. The processors characteristics, at the start, are shown in Table 2. The scheduling cost could be attributed to the software. After several tests, we have observed that the queue managers introduce the most important software delay. In our case, this delay is determined to be about 0.5 seconds.

Table 1: Network Characteristics. bw represents the bandwidth and srtt the smoothed round trip time.

C.E.	Network	
	bw (Mb/s)	srtt (ms)
Hermes	94.800	0.241
Tales	94.800	0.241
Popo	0.237	665.535

In this work, we perform several tests to verify the effect of the adaptive approach in the efficiency of QSS and ESS upon environment changes. For this purpose we compare the adaptive approach presented here, against the heuristic approach presented in [29]. The main difference between both approaches is that the first generates a new list of chunks when the environment changes, whereas the second generates a list of chunks that remains constant during the execution. Tests with 2000, 5000 and 10000 tasks are considered. During the tests we look for changes in the environment each 800 seconds (checktime). First, we compare the behaviour of the adaptive approach against the heuristic approach [29] in a static environment. That is, the environment will be static during all execution, but the adaptive approach will generate a new list of chunks each checktime. On the other hand, the heuristic approach will generate only one list of chunks when the execution starts.

Table 2: Processors Characteristics for each computer element (C.E.) The number of processors of each type is specified. The relative computing power (R.P.) is also included.

Computer Elements				
Hermes	Tales	Popo	Processor Type	R.P.
5	1		P4 3.0 GHz	1
	4		P4 2.8 GHz	0.585
3	3		P4 2.4 GHz	0.515
		4	AMD64 1.6 GHz	0.448

We have performed also several tests with a dynamic execution environment. In this case, the environment changes over time, while the algorithms behave as described above. Here, a processor has the possibility to increase, decrease or keep its performance in each checktime. We consider different processor performance variation rates for each test case. The variation rates considered are: 10%, 20%, 30%, 40%, and 50%. Moreover, we compare these results against the results obtained with the heuristic approach [29]. Each test is performed 10 times to obtain average results. In order to obtain a more representative average, we compute it using only the 8 most representative values. For that, we removed the two extreme values of each case (the best and the worst).

3. Results

First, we have performed different tests to compare the behaviour of the adaptive approach with a static environment during the execution. In this situation, we have performed different tests with QSS and ESS using the adaptive and the heuristic approaches to obtain the optimal list of chunks. The average results are collected in Figure 1. Here, we observe that in all cases the behaviour of both approaches is very similar. In particular, the maximum difference found is a 0.4%.

Now, we test both approaches in a dynamic environment. For that, we have performed several tests with different number of tasks and different percentage of possible variation rate in the processors performance. For QSS, the results are collected in Figure 2. In Figure 2, each test case is represented by a different line. The X axis shows the percentage that each processor can change each time that the processors information is updated. Thus, we observe that when variation rates increases, the adaptive approach becomes more efficient that the heuristic one. This effect is higher when the number of tasks increases. In these tests,

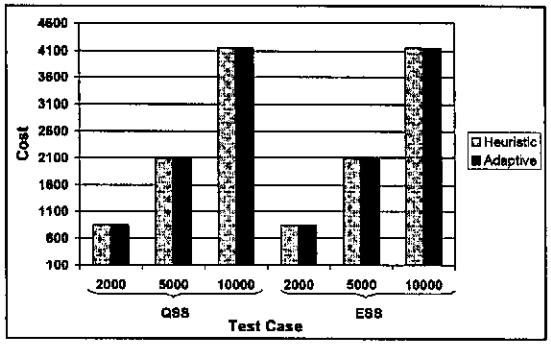

Figure 1: Average QSS and ESS cost for the heuristic and adaptive approaches. The cost is given in seconds.

the adaptive approach allows for a maximum improvement of 25% in the test with 10000 tasks and a 50% of variation rate.

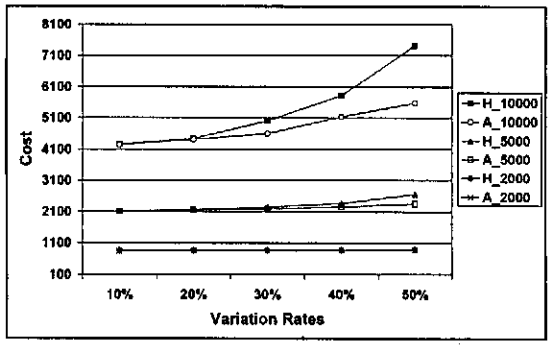

Figure 2: Average cost of QSS optimized using the Heuristic approach (H) and the Adaptive approach (A). The label of each test case is given by the approach and the number of tasks. The cost is given in seconds.

With respect to ESS, we have performed the same tests. Figure 3 shows graphically the behaviour of ESS using the adaptive and heuristic approaches. Nevertheless, the results for this algorithm are not specially good. Here, we observe that the adaptive approach is better in the cases with 2000 and 5000 tasks. In particular, the maximum difference in performance is 22%, which is found for the 5000 tasks test case and a variation rate of 50%. However, when the number of tasks increases and the variation rate is high (40% or 50%), the adaptive approach shows the worst performance. This behaviour arises because the chunks generated at the start of the execution are too large. It would not be a problem in an static environment, as shown in Figure 1, because the list of chunks is generated assuring a good load balance. However, when the environment changes strongly these large chunks generate an important load imbalance that can not be compensated with new lists of chunks.

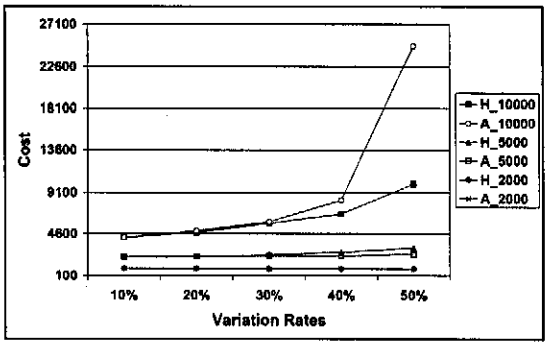

Figure 3: Average cost of ESS optimized using the Heuristic approach (H) and the Adaptive approach (A). The label of each test case is given by the approach and the number of tasks. The cost is given in seconds.

Finally, in Figure 4 we compare the behaviour of QSS and ESS, using the adaptive approach, in a dynamic environment. We can observe that when the number of tasks is small (2000 and 5000), both algorithms have a very similar behaviour. However, when the number of tasks increases, and the environment becomes more dynamic, QSS is more efficient. In particular, the performance of QSS using the adaptive approach is up to a 77% better in the test with 10000 tasks and a 50% of variation rate. Therefore, QSS is able to adapt more efficiently to changing environments when the number of tasks is high. The main reason for this difference is that QSS has more degrees of freedom, which implies a higher flexibility. Moreover, QSS generates more chunks than ESS, and they are smaller. Therefore, it is easier to obtain a better load balance in the succesive updates of the environment.

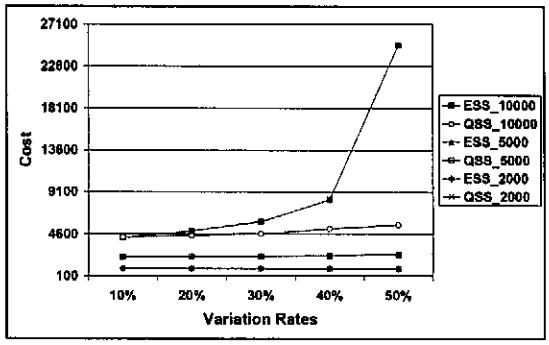

Figure 4: Average cost of QSS and ESS using the Adaptive approach (A). The label of each test case is given by the algorithm name and the number of tasks. The cost is given in seconds.

4. Conclusions

In this work, we present a new simulation-based adaptive approach to tackle the problem of scheduling a set of

independent tasks in a computational Grid. This work is focused to Grid environments where the available processors can change widely their performances. For this purpose, we have developed a new adaptive approach that allows an improvement in the efficiency of the QSS and in some cases of the ESS algorithms. We observe that using this new approach it is possible to obtain a better load balance in a dynamic environment. Nevertheless, the algorithm used by the adaptive approach should be flexible enough. QSS has shown to be very flexible and, within the adaptive approach, the performance improvement is around a 25% respect to a pure heuristic approach. On the other hand, the same is not true for ESS, whose behaviour degrades when the number of tasks is large and the environment is strongly dynamic. Finally, the overall time spent by the adaptive approach generating new lists of chunks, is higher than the time to obtain just one list. However, this time is not proportional to the number of lists generated. This is because the number of tasks decreases each time we generate a new list of chunks. In particular, for QSS the time necessary to finish the simulation in the adaptive approach is about 100 seconds. On the other hand, the time is around 40 seconds for the heuristic case. Finally, for ESS, both approaches spend more or less the same time.

In future works, our aim is to improve this approach including fault tolerance and resource discovering. In this way, we could reschedule lost tasks if a processor falls down or we could discover and use new processors. Thus, we would be considering another important issue in distributed environments.

Acknowledgment

This work has been cofinanced by FEDER funds, the Consejería de Educación y Ciencia de la Junta de Comunidades de Castilla-La Mancha (grant # PBI08-0008), and the fellowship associated to the grant # PBI-05-009. The Ministerio de Educación y Ciencia (grant # FIS2005-00293) and the Universidad de Castilla-La Mancha are also acknowledged.

References

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, San Francisco, CA: Morgan Kauffman Publishers, 1999.

[2] T. L. Casavant and J. G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing," *IEEE Trans. Softw. Eng.*, vol. 14, no. 2, pp. 141-154, 1988.

[3] D. J. Lilja, "Exploiting the Parallelism Available in Loops," *IEEE Computer*, vol. 27, vo. 2, pp. 13-26, 1994.

[4] P. J. Sokolowski, D. Grosu and C. Xu, "Analysis of Performance Behaviors of Grid Connected Clusters," in *Performance Evaluation of Parallel and Distributed Systems*, M. Ould-khaoua, and G. Min, Eds. Hauppauge, NY: Nova Science Publishers, 2006.

[5] S. Penmatsa, A. T. Chronopoulos, N. T. Karonis and B. Toonen, "Implementation of Distributed Loop Scheduling Schemes on the TeraGrid," *Proc. 21st IEEE Int. Parallel and Distrib. Proc. Symp. (IPDPS 2007), 4th High Performance Grid Computing Workshop*, 2007.

[6] J. Díaz, S. Reyes, A. Niño and C. Muñoz-Caro, "A Quadratic Self-Scheduling Algorithm for Heterogeneous Distributed Computing Systems," *Proc. 5th Int. Workshop Algorithms, Models and Tools for Parallel Computing Heterogeneous Networks* (HeteroPar '06), Barcelona, Spain, 2006, pp. 1-8.

[7] J. Díaz, S. Reyes, A. Niño, and C. Muñoz-Caro, "New Self-Scheduling Schemes for Internet-Based Grids of Computers," *1st Iberian Grid Infrastructure Conf.* (IBERGRID), Santiago de Compostela, Spain, 2007, pp. 184-195.

[8] J. Díaz, S. Reyes, A. Niño and C. Muñoz-Caro, "Un Algoritmo Autoplanificador Cuadrático para Clusters Heterogéneos de Computadores," *XVII Jornadas de Paralelismo*, Albacete, Spain, 2006, pp. 379-382.

[9] J. Díaz, S. Reyes, A. Niño, C. Muñoz-Caro, "Derivation of Self-Scheduling Algorithms for Heterogeneous Distributed Computer Systems: Application to Internet-based Grids of Computers," *Future Generation Computer Systems*, ElSevier, published online, DOI: 10.1016/j.future.2008.12.003.

[10] J. Yu and R. Buyya, "Workflow Schdeduling Algorithms for Grid Computing," Grid Computing and Distrib. Systems. Lab., Univ. Melbourne, Australia, Tech. Rep., GRIDS-TR-2007-10, May 2007.

[11] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," School of Computing, Queen's University, Kingston, Ontario, 2006.

[12] R. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen and R. Freund, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *J. Par. Dist. Com.*, vol. 61, no. 6, pp. 810-837, 2001.

[13] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Boston, MA: Addison-Wesley, 1989.

[14] M. Aggarwal, R. D. Kent and A. Ngom, "Genetic Algorithm Based Scheduler for Computational Grids," in *Proc. 19th Annu. Int. Symp. High Performance Computing Systems and Applications (HPCS'05)*, Guelph, Ontario Canada, 2005, pp.209-215.

[15] S. Kim, and J. B. Weissman, "A Genetic Algorithm Based Approach for Scheduling Decomposable Data Grid Applications," *Proc. 2004 Int. Conf. Parallel Processing* (ICPP'04), Montreal, Quebec Canada, 2004, pp. 406-413.

[16] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equations of State Calculations by Fast Computing Machines," *J. Chem. Phys.*, vol. 21, pp. 1087-1091, 1953.

[17] S. Fidanova, "Simulated Annealing for Grid Scheduling Problem," *Proc. IEEE John Vincent Atanasoff 2006 Int. Symp. Modern Computing*, 2006, pp. 41-45.

[18] A. YarKhan, and J. J. Dongarra, "Experiments with Scheduling Using Simulated Annealing in a Grid Environment," *Proc. 3rd Int. Workshop on Grid Computing*, Baltimore, MD, 2002, pp. 232-242.

[19] F. Glover and M. Laguna, *Tabu Search*, Boston, MA: Kluwer Academic Publishers, 1997.

[20] S. Benedict and V. Vasudevan, "Improving Scheduling of Scientific Workflows Using Tabu Search for Computational Grids," *Information Technology Journal*, vol. 7, no. 1, pp. 91-97, 2008.

[21] E. Huedo, R.S. Montero, I.M. Llorente, "Experiences on Adaptive Grid Scheduling of Parameter Sweep Applications," *Proc. 12th Euromicro Conference on Parallel,Distributed and Network-Based Processing*, 2004, pp. 28-33.

[22] F. Berman, R. Wolski, H. Casanova, "Adaptive Computing on the Grid Using AppLeS," *IEEE Trans. Parallel and Distributed Systems*, vol. 14, no. 4, pp. 369-382, 2003.

[23] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid," *4th Int. Conf. on High-Performance Computing in the Asia-Pacific Region*, vol. 1, 2000, pp. 1-7.

[24] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, J. Shalf,"The Cactus Worm: Experiments with dynamic resource discovery and allocation in a grid environment," *Int. J. of High Performance Computing Applications*, vol. 15, no. 4, pp. 345-358, 2001.

[25] I. Banicescu and Z. Liu, "Adaptive Factoring: A Dynamic Scheduling Method Tuned to the Rate of Weight Changes," *Proc. High Performance Computing Symp. (HPC 2000), The Society for Modeling and Simulation Int. (SCS)*, 2000, pp. 122-129.

[26] I. Banicescu, V. Velusamy, "Load balancing highly irregular computations with the adaptive factoring," *Proc. Int. Parallel and Distributed Processing Symp.*, 2002, pp. 87-98.

[27] M. Maheswaran, S. Ali, H.J. Siegal, D. Hensgen, R.F. Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems," *Proc. 8th Heterogeneous Computing Workshop*, 1999, pp. 30-44.

[28] H. Casanova, D. Zagorodnov, F. Berman, A. Legrand, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," *Proc. 9th Heterogeneous Computing Workshop*, 2000, pp. 349-353.

[29] J. Díaz, S. Reyes, C. Muñoz-Caro, and A. Niño, "A Heuristic Approach to Task Scheduling in Internet-based Grids of Computers," *2nd Int. Conf. on Advanced Engineering Computing and Applications in Sciences (ADVCOMP'08)*, Valencia, Spain, 2008, pp. 110-116.

[30] J. L. Hennessy and D. A. Patterson, *Computer Architecture. A Quantitative Approach*, 3th Ed., San Francisco, CA: Morgan Kaufmann Publishers, 2003.

[31] P. Karn, and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," *ACM Trans. Comput. Syst.*, vol. 9, no. 4, pp. 364-373, Nov. 1991.

[32] R. Buyya, *High Performance Cluster Computing: Architectures and Systems*. New Jersey: Prentice Hall, vol. 1, 1999.

[33] P.W.G. Morris, *The Management of Projects*, London : Thomas Telford, 1994.