A Fault Tolerant Adaptive Method for the Scheduling of Tasks in Dynamic Grids

Javier Díaz, Camelia Muñoz-Caro and Alfonso 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
E-mail:{javier.diaz, camelia.munoz, alfonso.nino}@uclm.es

***Abstract*—- An essential issue in distributed high-performance computing is how to allocate efficiently the workload among the processors. This is specially important in a computational Grid where its resources are heterogeneous and dynamic. Algorithms like Quadratic Self-Scheduling (QSS) and Exponential Self-Scheduling (ESS) are useful to obtain a good load balance, reducing the communication overhead. Here, it is proposed a fault tolerant adaptive approach to schedule tasks in dynamic Grid environments. The aim of this approach is to optimize the list of chunks that QSS and ESS generates, that is, the way to schedule the tasks. For that, when the environment changes, new optimal QSS and ESS parameters are obtained to schedule the remaining tasks in an optimal way, maintaining a good load balance. Moreover, failed tasks are rescheduled. The results show that the adaptive approach obtains a good performance of both QSS and ESS even in a highly dynamic environment.**

Keywords - Adaptive Scheduling; Heuristic Scheduling; Self-Scheduling Algorithms; Computational Grid.

I. INTRODUCTION

Computational Grids [15] 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 [8]). In particular, dynamic self-scheduling algorithms are extensively used in practical applications [19] [22] [24]. 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 shared-memory systems, see for instance [19]. This kind of algorithms divides the set of tasks into subsets (chunks), and allocates them among the processors (one chunk per processor). 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 [22] [24] [9] [10]. 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.

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 on heuristic algorithms [25] [14] 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. [4] presented three basic heuristics, based on Nature, for Grid scheduling. These are: Genetic Algorithms (GA) [17], Simulated Annealing (SA) [21] and Tabu Search (TS) [16].

Other way to tackle the problem of the unpredictable changing conditions of the Grid is using Adaptive Grid scheduling [18]. 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 [3], Nimrod/G [6], Cactus Worm [1] or GridWay [18]. Moreover, there are algorithms like adaptive factoring [2], Sufferage [20] or XSufferage [7].

We have previously proposed two new flexible self-scheduling algorithms called Quadratic Self-Scheduling (QSS) [9] and Exponential Self-Scheduling (ESS) [10] [11]. The first is based on a quadratic form for the chunks distribution function. 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. Moreover, in previous works [10] we compared our approaches with other self-scheduling algorithms in an actual Grid environment. Our approaches outperformed the previous ones, since they obtained better load balance and more reduction of the communication overhead. However, QSS and ESS depend on three and two parameters respectively, which determine their behavior. Therefore, it is necessary to select the most appropriate (optimal) values of these parameters for a given computational environment. In this way, we could obtain a good load balance and minimize the overall computation

time.

Previously, we presented a way to obtain optimal QSS and ESS parameters using a heuristic approach [12]. To such an end, we simulated the execution environment (a Computational Grid in our case). Using the simulation, we could obtain the computational time of each algorithm for a given value of its parameters. Therefore, it was possible to apply a 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 were the "best" ones for the environment where they were obtained. Therefore, if the environment characteristics change strongly, we could have a lack of efficiency. Thus, in [13] we introduced an early version of a new adaptive approach to schedule tasks in dynamic Grid environments. Using this approach, new optimal QSS and ESS parameters were obtained when the environment change. In this way, it was possible to maintain a good load balance. In [13], we only considered the variation of the processor's computing power to simulate a dynamic environment. However, in a real case both processors and clusters can appear or disappear.

In this paper, we present a fault tolerant adaptive approach to schedule tasks in dynamic Grid environments. Thus, the adaptive approach also tackles the case in which processors and clusters appear or disappear. In addition, if a processor or cluster disappears, their tasks are rescheduled. 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 work.

II. Methodology

In this work, the Quadratic Self-Scheduling (QSS) and Exponential Self-Scheduling (QSS) algorithms are used as basic scheduling strategies. QSS [9] [10] [11] 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 [9] 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 are 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) [10] [11]. ESS belongs to the family of algorithms based on 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,

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

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. In this case g(t)=-kC(t). Therefore,

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

where 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 [10],

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

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 [12], we consider that the function to minimize, the cost function (f), is the overall computation time needed to process all 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 arbitrary time units, atus) necessary to solve all tasks in the specified execution environment. The tasks are scheduled according to the chunk distribution of QSS or ESS. The chunk distributions depend on the optimal QSS or ESS parameters, which are given by the final value of s obtained by simulated annealing. Moreover, in each cluster is allocated as many chunks as processors available. The simulator is organized as follows. Each task has associated a value, from 1 to 10, which represents its duration (in atus). Task durations are randomly generated. As previously commented, QSS and ESS

allocate sets of tasks (chunks). Since all tasks are executed in a single processor, 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 queueing software is included as well. The resulting value represents the estimated completion time.

The execution environment represented in the simulation is an Internet-Based Grid of computers [15]. 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 atus, of transferring a file between two machines through the Internet. On the other hand, a computer element is typically a computer cluster [5]. 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 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 [13] we presented an adaptive approach to tackle this case. Nevertheless, in this work we have improved this previous approach, to consider a more realistic system in which processors or whole clusters can appear or disappear. Thus, we have created an application, which simulates the chunks execution (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
      if a processor or cluster disappear then
        if all tasks were not executed then
          7.- Add all them to the pool of tasks
        end_if
      end_if
      8.-Look for new clusters or processors
      9.-Obtain new optimal parameters
      10.-Generate new list of chunks
    end_if
  done_while
end_Function
```

The first step generates the simulated execution environment. After that, the optimal parameters of QSS or ESS algorithms are obtained using the heuristic approach [12] 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 [23] for the estimated completion time, in each processor, of the allocated chunks. Now, we update the execution environment characteristics and restore the saved execution status. In the case that a processor or a cluster disappears, we check if its tasks were all executed. If all tasks were not executed, we reschedule all of them (adding them to the pool of tasks). After this, we look for new clusters or processors. Since the environment has changed, we have to generate new optimal parameters for our self-scheduling algorithm. For that, we use the heuristic approach presented in [12]. 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 its assigned chunks. In this way, we obtain a better load balance. Once we have the new optimal parameters for the self-scheduling algorithms, we generate a new list of chunks and continue in the 4th step. The "while" loop finishes when all tasks are scheduled and executed.

The simulated execution environment used is initially composed by three clusters (Hermes, Tales, Popo), see Tables I and II. However, since the environment is dynamic, new clusters can appear as well as disappear. In particular, we have considered three additional clusters that can appear during the execution, see Tables I and II. Moreover, in each cluster, processors can appear or disappear. When a processor appears, its computer power is selected randomly between 0.3 and 1. We have subdivided the environment characteristics into three parts: network, processors and scheduling cost. After several actual tests, we have observed that the queue managers introduce the most important scheduling cost. In our case, this cost is taking to be about 0.5 atus. In Tables I and II, we collect the network and processors characteristics of each cluster.

TABLE I
NETWORK CHARACTERISTICS SPECIFIED USING ITS COST IN ARBITRARY TEMPORAL UNITS (ATUS).

Computer Elements					
Hermes	Tales	Popo	Edipo	Zenon	Aris
0.106	0.106	4.612	1.612	0.206	0.206

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. Tests with 10000 tasks are considered. During the tests we look for changes in the environment each 800 atus (checktime). This value is independent of the model and it could be any other. Here, a processor has the possibility

TABLE II
PROCESSORS CHARACTERISTICS FOR EACH COMPUTER ELEMENT. THE NUMBER OF PROCESSORS OF EACH TYPE IS SPECIFIED USING ITS RELATIVE COMPUTING POWER (R.P.).

	Computer Elements					
R.P.	Hermes	Tales	Popo	Edipo	Zenon	Aris
1	5	1	1	1	1	2
0.915						1
0.585		4				3
0.548				1		
0.515	3	3			2	2
0.448			4	1		
0.348				1		

to increase, decrease or keep its performance in each checktime. We consider that the processor performance variation rate is fixed to 10%. Previously, we studied the variation of this percentage [13]. On the other hand, both clusters and processors can appear or disappear during the execution in each checktime. We consider different probabilities in each test case. The probability that a processor appears or disappears is taking as 10, 20 or 30 %. In the case of the clusters, the probability is taking as 5, 10 or 15 %. In order to analyze the behaviour of the adaptive approach, we perform tests in which processors and clusters appear, disappear or both situations together. Each test is performed 20 times to obtain average results.

III. RESULTS

First, we have performed tests in which processors or clusters only disappear. Thus, it is possible to observe the influence of this situation in the behaviour of our algorithms. The average results are collected in Figure 1. We observe that in both cases, QSS and ESS, the performance is better when the probability of processors or clusters disappearance is low. In particular, clusters disappearance has the highest influence on the performance, because it implies that several processors disappear at one stroke. Moreover, we observe that ESS has a better performance than QSS. In particular, the maximum difference found is a 29%. The difference in the performance is due to the way each algorithm distributes the tasks. ESS generates a reduced list of chunks whose sizes are very large at first, decreasing quickly. On the other hand, QSS generates a longer list of chunks whose sizes are not so large at first and decrease slowly. Thus, scheduling the tasks with ESS, we could lose more tasks than using QSS, due to the size of its chunks. However, this way to schedule the tasks provides a better performance when there is a reduced number of processors, because it reduces the network overheads keeping the load balance. However, since QSS generates more chunks, there are more probabilities that a chunk fails.

On the other hand, we have performed tests in which processors or clusters only appear, see Figure 2. In this case, the behaviour of both algorithms is the opposite of the previous case. Here, the best situation is for a high probability of processors or clusters appearance, because we get more

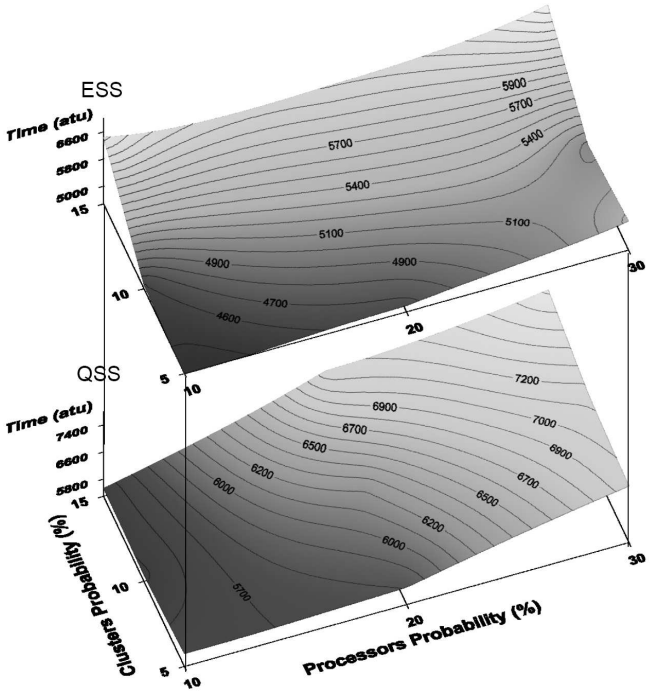

Fig. 1. Average ESS (up) and QSS (down) cost as a function of the processors and clusters disappearance probability. The temporal cost is given in atus.

processors where execute the tasks. Moreover, we observe that the QSS algorithm is up to a 12% better than ESS. This is because the previous disadvantage of QSS now is an advantage, since the higher number of chunks allows a better load balance of the workload when new processors appear.

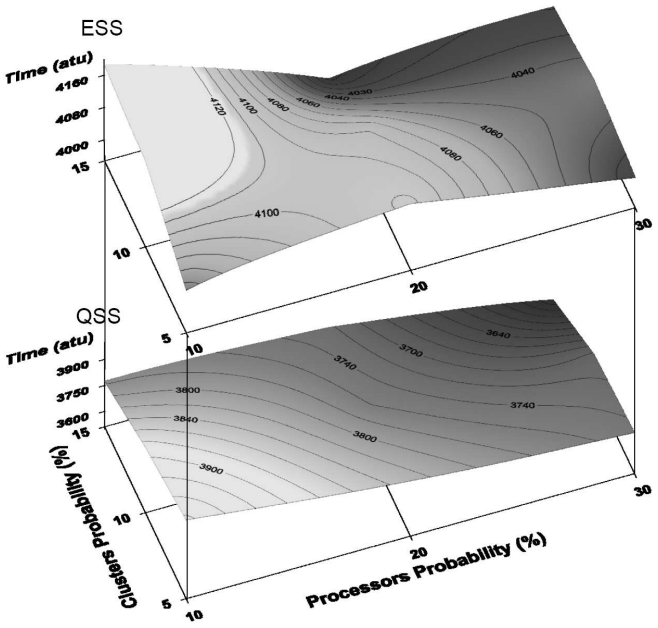

Fig. 2. Average ESS (up) and QSS (down) cost as a function of the processors and clusters appearance probability. The temporal cost is given in atus.

Finally, we have tested the behaviour of QSS and ESS

when processors or clusters appear and disappear. Here, the probability of appearance or disappearance is the same. In each checktime, we check if a processor or cluster disappears and then if a processor or cluster appears. Figure 3 shows graphically the behaviour of QSS and ESS. We observe that the best behaviour of both algorithms is in the middle of each figure. That is the expected behaviour, since this case can be considered as the combination of the two previous ones. Here, ESS provides better performance than QSS. The maximum difference found is a 18%, although in some cases both algorithms have very similar performance.

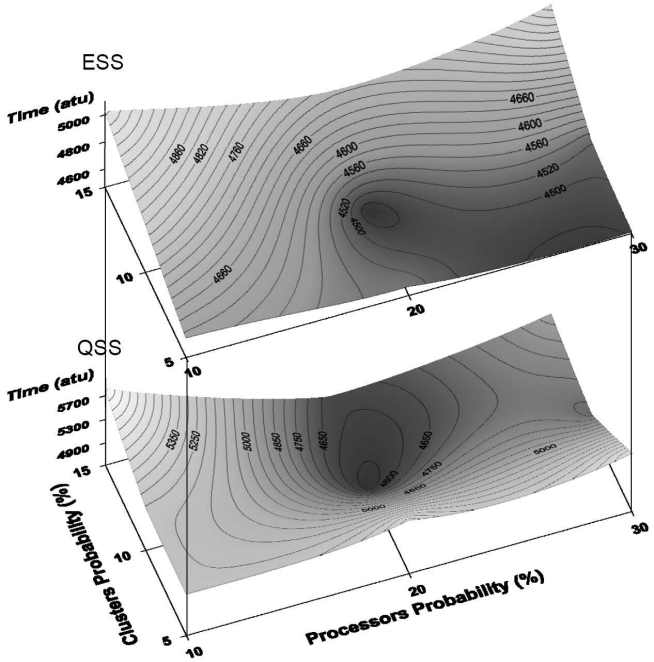

Fig. 3. Average ESS (up) and QSS (down) cost as a function of the processors and clusters appearance/disappearance probability. The temporal cost is given in atus.

These results show that the adaptive approach not only supports fault tolerance, but also allows adaptability to changing Grid environments. Thus, the execution time of this approach when only the computer power of the processors changes (semi-static environment) is 4242 and 4258 atus for the QSS and ESS algorithms, respectively. Comparing these results with those obtained when processors or clusters appear and disappear, we find that in this last case the execution time has a maximum delay of 27% and 15% for QSS and ESS respectively. This difference is found for the case of 10% processors probability and 15% clusters probability.

IV. Conclusion

In this work, we present a fault tolerant adaptive approach to schedule tasks in dynamic Grid environments. Here, we consider environments in which the computational power of a processor can change, new processors or clusters can be discovered, and processors or clusters can disappear. In the last case, the failed tasks are rescheduled. We have seen that in a dynamic environment where processors appear and disappear, the behaviour of QSS and ESS is different, due to the way the tasks are distributed into the chunks to be scheduled. Thus, we observe that ESS has better performance when the number of processors is reduced. On the other hand, QSS is better when the number of processors increases. In the case where processors and clusters appear and disappear, ESS is also better than QSS. Therefore, the adaptive approach obtains a good performance of the algorithms even in a highly dynamic environment. In this situation, we find a maximum delay, respect to the semi-static environment, of a 27% and 15% for QSS and ESS, respectively.

In future works, our aim is to apply this approach to schedule tasks in an actual Grid environment. In this way, we can test if the adaptive approach is useful in an actual case.

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 Universidad de Castilla-La Mancha is also acknowledged.

References

[1] 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.

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

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

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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, vol. 25, no. 6, pp. 617-626, 2009.

[12] 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.

[13] J. Díaz, C. Muñoz-Caro, and A. Niño, "An Adaptive Approach to Task Scheduling Optimization in Dynamic Grid Environments," *The 2009 Int. Conf. on Grid Computing and Applications*, Las Vegas, USA, 2009.

[14] 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.

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

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

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

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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, 2007.