A Heuristic Approach to the Allocation of Different Workloads in Computational Grid Environments

Javier Díaz, Sebastián Reyes, 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,sebastian.reyes,camelia.munoz,alfonso.nino}@uclm.es

Abstract

Self-scheduling algorithms can achieve a good balance between workload and communication overhead in computational systems. In particular, Quadratic Self-Scheduling (QSS) and Exponential Self-Scheduling (ESS) are flexible enough to adapt to distributed systems. Thus, they are of interest for application in Internet-based Grids of computers. We have tackled the problem of scheduling a set of independent tasks in a computational Grid using a simulator and a heuristic approach based in simulated annealing. To test this approach, we have considered different workload distributions. We find that the optimal Simulated Annealing (SA) results permit to reduce the overall computing time of a set of tasks up to a 16%, with respect to results obtained with previous values of the parameters experimentally determined. In addition, the time to obtain the SA optimized parameters by simulation is negligible compared with that needed using experimental measures. Moreover, after the optimization, the heuristic approach provides equivalent performance for different workloads (random, increasing, decreasing) and different number of tasks. These results show the high adaptability of the QSS and ESS self-scheduling algorithms, which can be fully exploited thanks to the heuristic approach here presented.

Keyword: Self-Scheduling Algorithms, Heuristic Scheduling, Computational Grid.

1. Introduction

A computational Grid [14] is a hardware and software infrastructure providing dependable, consistent, and pervasive access to resources among different administrative domains. The objective is to enable the sharing of these resources in a 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 [6]). 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 [24]. Self-scheduling algorithms divides the set of tasks into subsets (chunks), and allocates them among the processors. In this way overheads are reduced. However, the performance of a self-scheduling algorithm is not independent of the workload distribution. The workload represents the tasks duration distribution in a problem. Figure 1 shows the four possible cases: uniform workload, increasing workload, decreasing workload, and random workload. They have a direct influence in the performance of scheduling algorithms. In general, the random case is the most difficult to schedule, since the duration of the different tasks is not known beforehand. However, the increasing workload can be optimally scheduled using a decreasing chunk distribution function. This is because it assigns a large number of small tasks at first, and a few number of big tasks at the end, trying to guarantee a good load balance. On the other hand, the decreasing workload can be scheduled efficiently using an increasing chunk distribution function.

Self-scheduling algorithms have been tested successfully in distributed memory multiprocessor systems and heterogeneous clusters [3][7][17][22][27][31]. In addition, some works about their performance on Grid connected clusters

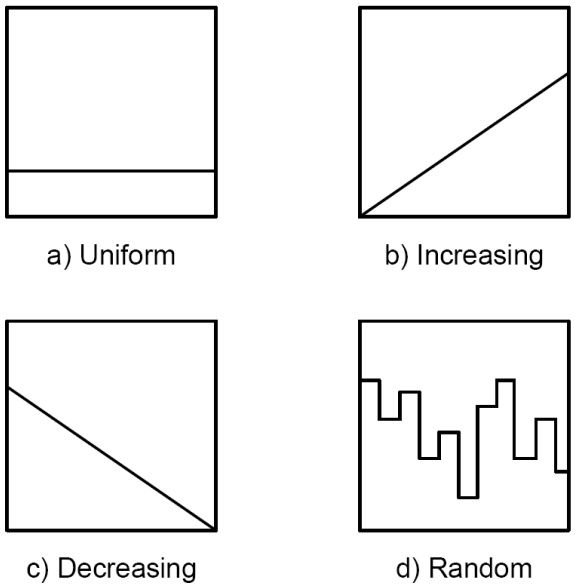

Figure 1. The four different workloads. a) uniform workload, b) increasing workload, c) decreasing workload, d) random workload

of computers have been reported [8][9][26][29]. Thus, although self-scheduling algorithms are derived for homogeneous systems, in principle, they can be applied to heterogeneous ones, such as computational Grids [14]. However, they could be not enough flexible (they may have not enough degrees of freedom) to adapt efficiently to a heterogeneous environment. In this sense, we have previously proposed two new flexible self-scheduling algorithms called Quadratic Self-Scheduling (QSS) [7][8] and Exponential Self-Scheduling (ESS) [9][10]. 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 through two parameters. Moreover, in previous works [9][10] 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 [9].

However, a computational Grid is made up of a large number of independent resource providers and consumers, which are running concurrently, changing dynamically, and interacting with each other. Due to these environment characteristics, new approaches such as those based in heuristic algorithms [20][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 [4] 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) [16] 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 [2][23][30]. A genetic algorithm combines exploitation of best solutions from past searches with the exploration of new regions of the solution space. A genetic algorithm for scheduling problems can be organized as follows [19]. First, an initial population is necessary, which can be generated by other heuristic algorithm. A population is a set of “chromosomes” where each represents a possible solution. A solution is a mapping sequence between tasks and machines. The “chromosomes” are evaluated and a fitness value is associated with each. The fitness value indicates how well the individual is compared with others in the population. Next, the population evolves, that is, a new generation is obtained by using the genetic operators, namely selection, crossover and mutation [16]. Finally, the chromosomes from this modified population are evaluated again. This completes one iteration of the GA. The GA stops when a predefined number of iterations are reached or all chromosomes converge to the same mapping.

Another heuristic algorithm is Simulated Annealing (SA) [25]. 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. That is, at high temperatures atoms or molecules move freely with respect to one another. If the system is cooled slowly, thermal mobility is lost and the atoms or molecules can line themselves up and form a pure crystal that is completely ordered. This crystal is the state of minimum energy for this system. Here, we focus in SA for scheduling purposes. This approach has been tested in different environments like computational Grids [13][32]. SA is organized in several steps. First, a simulated annealing algorithm needs an initial solution, which is constructed by assigning at random a resource to each task. The annealing process runs through a number of iterations at each "temperature" to sample the search space. At each iteration, it generates a new solution by applying a random change on the current solution. Whether or not the new solution is accepted as a current solution is determined by the Metropolis criteria [25]. Once a specified number of iterations have

been completed, the system is in "equilibrium", i.e., a population has been generated obeying a Boltzmann statistical distribution. Then, the temperature is decreased at a specified rate. The process is repeated until the lowest allowed temperature has been reached. During this process, the algorithm keeps the best solution found, returning the last value as the final optimal solution.

In Tabu Search (TS) [12][15] 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. This is one of the most important features of the algorithm. The algorithm starts from an initial solution, typically a random one. At any iteration it has to find a new solution by making local movements over the current solution. The next solution is the best among all (or a subset of) possible solutions in the neighborhood. To carry out the exploration process, recently visited solutions should be avoided (tabu list). Therefore, once a solution is visited the movement from which it was obtained is considered tabu. Note that the neighborhood of the solutions will be changing along the exploration. So, in a certain sense we have a dynamic neighborhood. Typically, there are two kinds of tabu lists. On one hand, a long term memory maintains the history through all the exploration process as a whole. On the other, a short term memory keeps the most recently visited tabu movements. A movement with a tabu status (tabu movement) is avoided unless it satisfies certain aspiration conditions. This prevents falling into local minima. There are two kinds of stopping conditions in Tabu Search. The first relates to the Tabu Search as a whole (when the algorithm finishes). The second is a stopping condition over the search of the best among all solutions in the neighborhood.

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. In previous studies, we obtained the best parameters from experimental measures on an actual system [8][9]. However, this is a slow and hard process that we would repeat each time the execution environment changes. In these conditions, the systematic exploration of the parameter space when several (more than two) parameters do exist is simply unmanageable.

Previously, we presented in [1] a way to obtain optimal QSS and ESS parameters using a heuristic approach. To such an end, we simulated the execution environment (a computational Grid in our case). So, 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).

Until now, we have tested these algorithms using a random workload distribution. Although, a priori, this is the most difficult case to schedule, it is interesting to observe the behaviour of our algorithms when they have to schedule other workload distributions. We have to consider that self-scheduling algorithms distribute the tasks into chunks in a decreasing way and therefore the load balance can be different depending on the task duration distributions. Considering as starting point the heuristic approach mentioned before, we study here its scheduling performance for the different workload distributions, see Figure 1.

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 in the different workload distributions considered. Section 3 presents and interprets the results found in the optimization process for each workload distribution. 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 [8][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 [8] 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. These two cases are defined by

$$\begin{aligned} Case\ a \Rightarrow C_{N/2} > \frac{C_0 + C_N}{2} \ \Rightarrow \frac{d^2C(t)}{dt^2} < 0 \\ Case\ b \Rightarrow C_{N/2} < \frac{C_0 + C_N}{2} \ \Rightarrow \frac{d^2C(t)}{dt^2} > 0 \end{aligned} \tag{5}$$

The two cases are compared with the linear model in Figure 2. As we can see, case a) is a function concave down. Here $N_q < N_l$, where the q and l subscripts are used to distinguish the quadratic from the linear model, respectively. This smaller number of chunks in the QSS algorithm translates in a smaller communication overhead to the expense of higher initial chunk sizes, see Figure 2. On the other hand, it is possible to invert this behaviour, as shown by the dashed line in Figure 2 (case b), where the function is concave up. Here, chunk sizes are smaller than in the linear case (almost all the way) to the expense of a higher number of chunks, $N_q' > N_l$. In this case, we have a better load balance, but the communication overhead increases. Therefore, the QSS algorithm can be tuned to optimize the ratio of load balance to overhead by selecting an appropriate value of the $C_{N/2}$ coefficient.

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) [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,

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

Equation (6) 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,

$$\frac{dC(t)}{dt} = -kC(t) \tag{7}$$

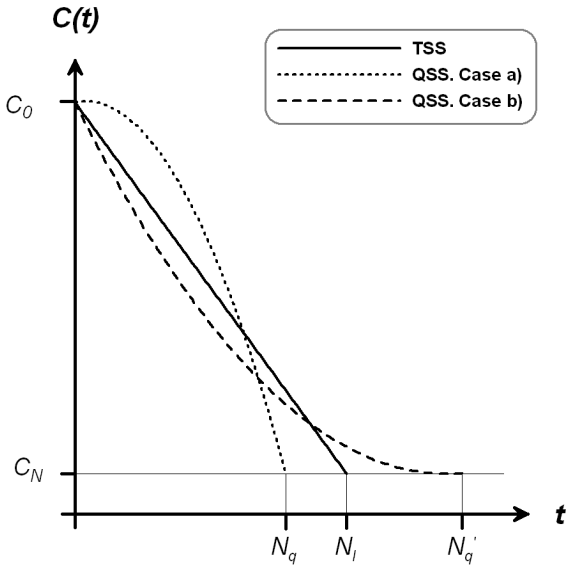

Figure 2. Evolution of the $C(t)$ function for the linear case (TSS) and QSS algorithms as a function of the number of chunks, t.

Here, k is a parameter and t represents the t-th chunk assigned to a processor. Equation (7) can be integrated by separation of variables yielding [9],

$$C(t) = C_0 e^{-kt} \tag{8}$$

Equation (8) 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, 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 corresponding pseudocode is shown in Chart 1.

Here, T is the system's "temperature" defined as a given value of the cost function, f(s). This is initialized to a high value. Symbols s and s' represent sets of the scheduling algorithm parameters. Three parameters are needed for QSS, and two for ESS. The f(s) function represents the cost of the s solution, which is initialized to a high value. As cooling schedule, we use an exponential approach $T_{n+1} = rT_n$, with $r = 0.989$ [28]. The total number of iterations performed for each temperature (equilibration) is given by L. In our case, after a previous calibration, we have fixed L to 200 iterations. Finally, FROZEN is the lowest allowed temperature for the system. We select a very small value for FROZEN, $12\text{x}10^{-11}$, which has shown to give consistent results in SA calibration tests.

Chart 1. Simulated Annealing Pseudocode

```
Function SimulatedAnnealing()
begin
  T := initial "temperature"
  s := initial parameters, S0
  repeat
    for i := 1 to L do begin
      generates new parameters s'
      if f(s') < f(s) then
        s := s'
      else begin
        x := f(s')-f(s);
        if e^{-x/T} > random(0,1) then
          s:= s'
      end
    end
    T := r T
  until FROZEN
  return s
end
```

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 in Chart 1. 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 except when we want to test a uniform workload. In this case, the duration of each task is 5. On the other hand, in the increasing and decreasing cases the task durations are shorted in the required order. As previously commented, QSS and ESS allocate sets of tasks (chunks). So, 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 [14]. 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 [18]. To such an end, we consider actual measures of the bandwidth (bw) and of the smoothed round trip time (srtt) [21]. 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 [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 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.

The simulated execution environment used is a replica of the computational Grid considered in [9]. In Figure 3, we can see the execution environment. 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 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	
Hermes	bw = 94.8 Mb/s	srtt = 241 μs
Tales	bw = 94.8 Mb/s	srtt = 241 μs
Popo	bw = 243 Kb/s	srtt = 665.535 ms

In this work, we perform several tests to verify the effect of SA optimized parameters in the efficiency of QSS and ESS. Tests with 1000, 2000, 5000 and 10000 tasks are considered. To determine the influence of the physical chunk size, these tests have been performed twice using a random workload. In the first case, we consider a physical chunk size of 1 Mb. In the second, we increase the chunk size to 10 Mb. The optimal values for the QSS and ESS parameters obtained by SA are used to compare the behavior of QSS and ESS. Moreover, we compare these results against

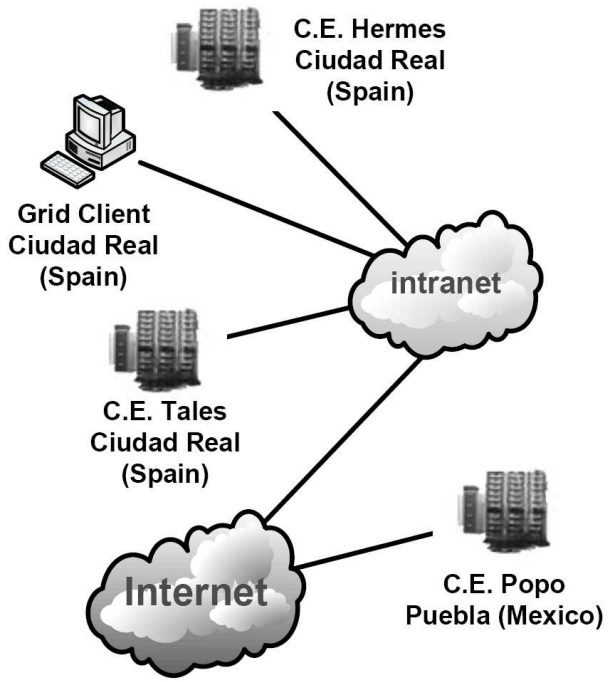

Figure 3. Execution Environment Simulated for the different tests: Internet based Grid of Computers.

the results obtained using the parameters experimentally determined for QSS and ESS in [9]. Finally, we compare the behaviour of the algorithms in the four workload distributions collected in Figure 1. In each case, new optimal QSS and ESS parameters are obtained. On the other hand, the increasing and decreasing workloads are obtained generating random tasks, and sorting them in the correct way. Each test is performed 100 times to obtain average results and determine standard deviations.

3. Results

First, we have performed different tests, using a random workload, to obtain the SA optimal parameters for QSS and ESS. Table 3 collects the results of the different tests, including the standard deviation (σ). In all cases, we observe a small σ. This implies that all the simulated values of the cost function are close to the average. Therefore, they can be considered very reliable. The main source for σ is the random generation of task durations in each simulation. Table 3 shows that the cost function increases linearly with the number of tasks. Comparing QSS and ESS, we see that both exhibit a similar performance. This result agrees with the data obtained in the experimental tests performed in [9].

To analyse the effect of the chunk sizes in the performance, we have repeated the simulations using a chunk size of 10Mb. Table 4 collects the results. We observe that the new values of the cost function, although higher as they must be, are very similar to those of Table 3. These results show that the present scheduling heuristic approach can compensate efficiently for changes of the network performance in a Grid system.

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

C.E.	Processors		
Hermes	**5** x P4 3.0 GHz		**3** x P4 2.4 GHz
R.P.	1		0.515
Tales	**1** x P4 3.0 GHz	**4** x P4 2.8 GHz	**3** x P4 2.4 GHz
R.P.	1	0.585	0.515
Popo	**4** x AMD64 1.6 GHz		
R.P.	0.448		

Table 3. Average cost function (cost) and standard deviation (σ) for QSS and ESS as a function of different number of tasks, in columns. The cost represents the simulated time, in seconds, necessary to solve all tasks. A physical chunk size of 1Mb is used.

QSS	1000	2000	5000	10000
Cost	428.830	846.191	2098.992	4179.476
σ	6.576	8.194	12.672	19.489
ESS	1000	2000	5000	10000
Cost	428.924	846.571	2098.921	4183.082
σ	6.213	8.817	15.341	17.586

Using SA, we obtained the minimum cost, total computing time, of QSS and ESS for the different test cases when the workload distribution is random. It would be interesting to compare these results against those obtained by using the QSS and ESS parameters experimentally determined in an actual Grid, see [9]. For QSS, these parameters were $C_0 = I/2P$, $\delta = 3$ and $C_N = 2$, where I is the total number of tasks and P is the number of processors. With these values, we have simulated the behavior of QSS obtaining the cost function, total computing time, of each test

Table 4. Average cost function (cost) and standard deviation (σ) for QSS and ESS as a function of different number of tasks, in columns. The cost represents the simulated time, in seconds, necessary to solve all tasks. A physical chunk size of 10Mb is used.

QSS	1000	2000	5000	10000
Cost	441.256	862.855	2120.979	4211.108
σ	7.183	8.399	15.672	19.253
ESS	1000	2000	5000	10000
Cost	445.563	870.002	2129.203	4211.697
σ	6.143	9.569	16.597	22.767

case. Since we have shown previously that the chunk size is not very significant, we only use a physical chunk size of 1Mb. Figure 4 collects the QSS results. We observe that the cost associated to the SA optimized parameters, "QSS SA", is always lower than that associated to the experimentally optimized QSS, "QSS E". In particular, we find that SA allows for an improvement between 3% and 6%. This is an interesting result, since SA obtains the most appropriate parameter values in 12 to 60 seconds, whereas the experimental values need a lengthy time consuming procedure on the actual Grid system [9].

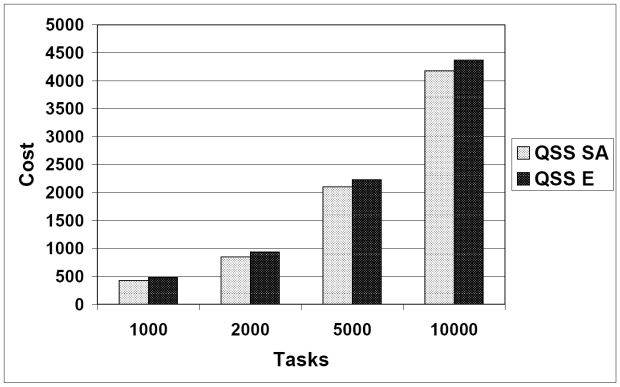

Figure 4. Comparison between the cost of QSS optimized using SA (QSS SA) and the cost of QSS optimized experimentally (QSS E). The cost is given in seconds.

With respect to ESS, the experimental parameters found in [9] are $Co = I/2P$ and $k = 0.017$. As in the QSS case, we have simulated the ESS behavior in our test cases using these values. Figure 5 shows graphically a comparison of the ESS results. We can appreciate that the cost of ESS with SA, "ESS SA", is always lower than that for the experimental parameters, "ESS E". In this case, simulated annealing gives us an improvement between 9% and 12%. Now, only 15 to 45 seconds are needed by SA to obtain the optimal results.

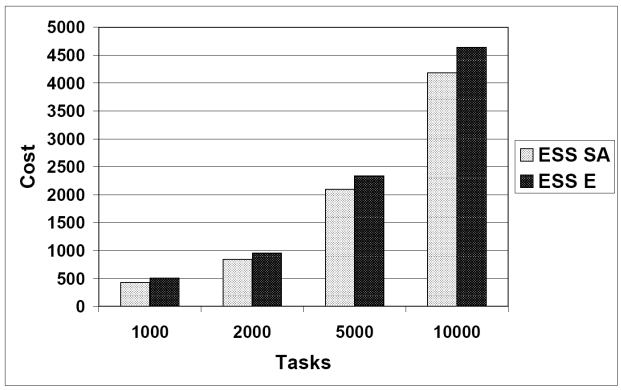

Figure 5. Comparison between the cost of ESS optimized using SA (ESS SA) and the cost of ESS optimized experimentally (ESS E). The cost is given in seconds.

The previous experiments were done using a random workload distribution. Usually, this kind of workload is the most difficult to handle, since the behaviour is unpredictable and therefore, it is more difficult to guarantee a good load balance. However, it is important to assess the behaviour of our algorithms when they have to schedule the remaining workloads: uniform, increasing, and decreasing (see Figure1). We have seen that the scheduling results of these workloads follows the results of the random one, with the QSS and ESS algorithms outperforming the rest of them. Therefore, here it is only exposed the results of the tests performed for QSS and ESS.

First, we have performed the tests using the parameters obtained experimentally. Thus, Figure 6 and Figure 7 show graphically the behaviour of QSS and ESS algorithms with the different workloads and the different test cases (number of tasks represented in x axis). The y axis represents the computational cost in seconds. We can observe that both QSS and ESS have a similar behaviour when scheduling different kinds of workloads. In both algorithms, the worst case corresponds to the increasing workload distribution. This is because self-scheduling algorithms distribute the tasks into chunks in a decreasing way. Therefore, too large tasks at the end tend to cause load imbalance. On the other hand, the decreasing workload have a good performance, even better than the random case. Here, there is too much workload at the start, but the algorithms can obtain a good load balance because they have enough time to

guarantee it. Nevertheless, the maximum difference in the execution time of the workloads is around a 12%. Finally, the uniform case is the less time consuming (between a 9% and a 18 % less). This is because all tasks have the same computational cost and because an uniforme workload is easier to schedule.

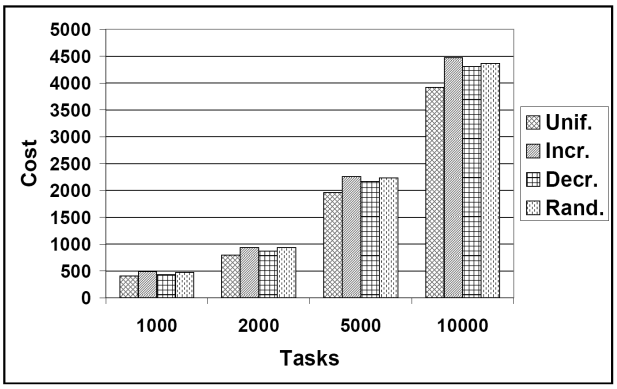

Figure 6. Behaviour of "QSS E" for different workload distributions. The QSS parameters were optimized experimentally (QSS E). The cost is given in seconds.

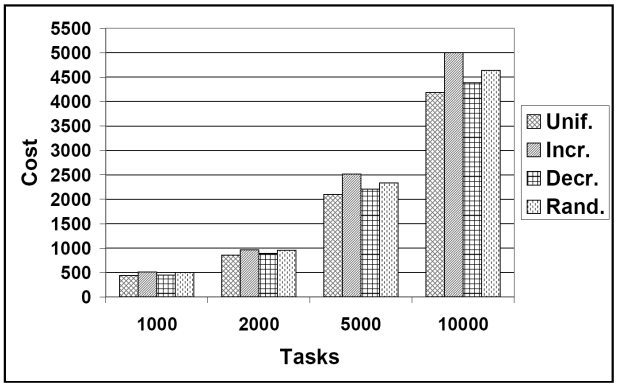

Figure 7. Behaviour of "ESS E" for different workload distributions. The ESS parameters were optimized experimentally (ESS E). The cost is given in seconds.

Now, for each case, we consider the behaviour of QSS and ESS after optimizing their parameters with the heuristic approach. Figure 8 and Figure 9 show the behaviour of both algorithms. We can observe that the uniform workload is again the less time consuming (around a 9%). On the other hand, the other workloads are executed more or less in the same time. So much so that the maximum difference in the execution time, among these workloads, is 1.3%. Therefore, even for different workloads and different number of tasks the heuristic approach provides equivalent performance.

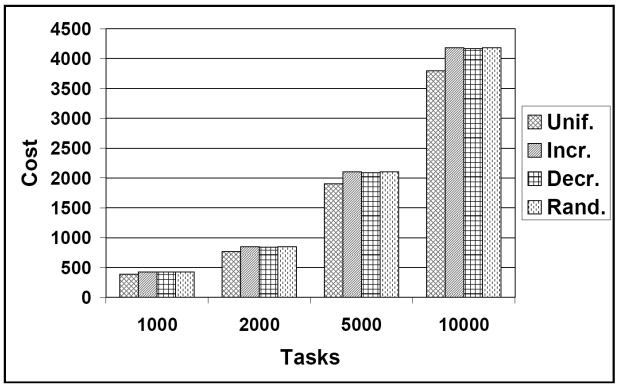

Figure 8. Behaviour of "QSS SA" for different workload distributions. The QSS parameters were optimized with SA (QSS SA). The cost is given in seconds.

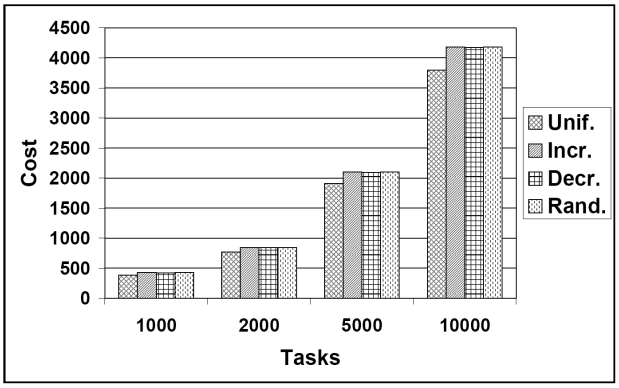

Figure 9. Behaviour of "ESS SA" for different workload distributions. The ESS parameters were optimized with SA (ESS SA). The cost is given in seconds.

4. Conclusion and Future Work

We have tackled the problem of scheduling a set of independent tasks in a computational Grid using a simulator and a heuristic approach based in SA. Using the simulator we can model the behavior of scheduling algorithms in an actual environment, but in a much shorter time than in an experimental study. We consider several test cases formed by several thousand tasks. These test cases can be subdivided in four groups depending on their workload distri-

butions. Thus, random, uniform, increasing and decreasing workload distributions are considered. A comparison like this is important, because there are situations in which the duration of the tasks is not known. Therefore, the tasks can not be sorted to optimize the scheduling. Working with the previously proposed QSS and ESS self-scheduling algorithms, we observe that optimizing their parameters using SA permits to reduce the overall computing time up to a 16%. This maximum difference is found in the increasing workload case. Moreover, after this optimization, we observe that the increasing, decreasing and random workloads have, more or less, the same overall execution time. These results show the high adaptability of our self-scheduling algorithms, which can be fully exploited thanks to the heuristic approach here presented.

The heuristic approach allows a reduction of the overall computing time. In addition, the time needed to obtain the optimal SA parameters for QSS and ESS, in the simulated environment, is negligible compared with the time needed for an experimental calibration in an actual Grid system. Furthermore, SA can optimize all the parameters in the scheduling algorithms, despite its number. In the general case, this is not possible using experimental measures. The test cases also show that the present heuristic approach is very efficient. In fact, we observe a simple linear increase of the execution time with the problem size.

Several enhancements can be devised for this preliminary study of a SA Grid scheduler. First, in the present study the characteristics of the Grid environment are considered static. A logical extension would be the scheduler to check, as a function of time, the state of the different network links and the performance of the available processors in the execution environment. Then, the simulated annealing procedure would obtain the most appropriate parameters for this specific behavior of the environment. Second, in the present work the chunk size is considered constant. This is an acceptable approach for testing and comparing the efficiency of scheduling algorithms in similar conditions. However, the situation is different in actual computations, where there are different physical chunk sizes. These sizes depend on the number of tasks making up the chunk. Considering this effect will permit to describe the different transfer costs of different chunks.

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. The authors wish also to thank the Facultad de Ciencias Químicas and the Laboratorio de Química Teórica of the Universidad Autónoma de Puebla (Mexico), for the use of the Popo cluster.

References

[1] 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 2008)*, Valencia, Spain, 2008, pp. 110-116.

[2] 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.

[3] F. Berman, "High-performance schedulers," in *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, Eds. San Fransisco, CA: Morgan-Kaufmann, 1999, pp. 279-309.

[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] 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.

[7] 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.

[8] 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.

[9] 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.

[10] J. Díaz, S. Reyes, A. Niño, C. Muñoz-Caro, "Nuevas Familias de Algoritmos de Self-Scheduling para la Planificación de Tareas en Grids de Computadores," *XVIII Jornadas de Paralelismo*, Zaragoza, Spain, 2007, pp. 423-430.

[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] I. D. Falco, R. D. Balio, E. Tarantino and R. Vaccaro, "Improving search by incorporating evolution principles in parallel tabu search," *IEEE Conf. Evolutionary Computation*, vol. 2, 1994, pp. 823-828.

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

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

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

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

[17] B. Hamidzadeh, D. J. Lilja and Y. Atif, "Dynamic Scheduling Techniques for Heterogeneous Computing Systems," *Concurr.: Pract. Exp*, vol. 7, pp. 633-652, 1995.

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

[19] E. S. H. Hou, N. Ansari and H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 2, pp. 113-120, Feb. 1994.

[20] 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.

[21] 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.

[22] T. H. Kim, and J. M. Purtilo, "Load Balancing for Parallel Loops in Workstation Clusters," *Proc. Int. Conf. Parallel Processing*, vol. 3, 1996, pp. 182-189.

[23] 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.

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

[25] 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.

[26] 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.

[27] T. Philip, and C. R. Das, "Evaluation of Loop Scheduling Algorithms on Distributed Memory Systems," *Proc. Int. Conf. Parallel and Distrib. Computing Systems*, Washinton DC, 1997.

[28] P. Salamon, P. Sibani, R. Frost, *Facts, Conjectures, and Improvements for Simulated Annealing*, Philadelphia: SIAM Monographs Mathematical Modeling and Computation, 2002.

[29] 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.

[30] S. Song, Y. Kwok and K. Hwang, "Security-Driven Heuristics and A Fast Genetic Algorithm for Trusted Grid Job Scheduling," *Proc. 19th IEEE Int. Parallel and Distrib. Proc. Symp.* (IPDPS'05), Denver, Colorado USA, 2005, pp. 65-74.

[31] C.-T. Yang, and S.-C. Chang, "A Parallel Loop Self-Scheduling on Extremely Heterogeneous PC Clusters," *9th Workshop Compiler Techniques for High-Performance Computing*, Academia Sinica, Taiwan, 2003.

[32] 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.